RT–11, RSTS/E FORTRAN IV

User's Guide

Order No. AA-5749B-TC

June 1980

This document describes the operating procedures for the FORTRAN IV Compiler and Object Time System (OTS) for the operating systems listed below. In conjunction with the *PDP-11 FORTRAN Language Reference Manual* (AA-1855D-TC), this document provides the information required to write and run a FORTRAN IV program under the operating systems listed below.

SUPERSESSION/UPDATE INFORMATION: This document completely supersedes the document of the same name, Order No. DEC-11-LRRUB-A-D.

OPERATING SYSTEM AND VERSION: RT-11 V04
RSTS/E V07

SOFTWARE VERSION: FORTRAN IV V02.5

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation · maynard, massachusetts

First Printing, December 1975
Revised: September 1977
Revised: June 1980

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10
VAX	VMS	SBI
DECnet	IAS	PDT
DATATRIEVE	TRAX	

CONTENTS

			Page
PREFACE			vii
CHAPTER	1	OPERATING PROCEDURES	1-1
	1.1	USING THE FORTRAN IV SYSTEM	1-1
	1.1.1	Compiler-Generated Code	1-1
	1.1.1.1	Code Options	1-2
	1.1.1.2	Code Selection and Error Messages	1-2
	1.1.2	File Name Specifications	1-3
	1.1.3	Locating a File	1-7
	1.2	RUNNING THE FORTRAN IV COMPILER	1-7
	1.2.1	Under RT-11 (with Keyboard Monitor Commands)	1-7
	1.2.1.1	Under RSTS/E	1-8
	1.2.2	Compiler Options (Switches)	1-9
	1.2.3	Listing Formats	1-11
	1.2.3.1	Options Listing	1-15
	1.2.3.2	Source Listing	1-15
	1.2.3.3	Storage Map Listing	1-15
	1.2.3.4	Generated Code Listing	1-15
	1.2.3.5	Compilation Statistics	1-16
	1.2.4	Compiler Memory Requirements	1-16
	1.2.4.1	Compiler Memory Requirements Under RT-11	1-16
	1.2.4.2	Compiler Memory Requirements Under RSTS/E	1-16
	1.3	LINKING PROCEDURES	1-17
	1.3.1	Linking Under RT-11	1-17
	1.3.2	Linking Under RSTS/E	1-20
	1.4	LIBRARY USAGE	1-24
	1.4.1	Overlay Usage	1-25
	1.4.2	Extended Memory Overlays (RT-11 Only)	1-27
	1.4.3	Stand-Alone FORTRAN	1-28
	1.4.4	FORTRAN Programs Run as VIRTUAL Jobs (RT-11 Only)	1-28
	1.4.5	Unformatted Byte I/O	1-29
	1.4.6	Smaller Execution-Time Programs	1-30
	1.5	EXECUTION PROCEDURES	1-30
	1.5.1	Execution Under RT-11	1-30
	1.5.2	Execution Under RSTS/E	1-30
	1.6	DEBUGGING A FORTRAN IV PROGRAM	1-31
CHAPTER	2	FORTRAN IV OPERATING ENVIRONMENT	2-1
	2.1	FORTRAN IV OBJECT-TIME SYSTEM	2-1
	2.2	OBJECT CODE	2-1
	2.2.1	Processor-Defined Functions	2-3
	2.2.2	VIRTUAL Array Options (RT-11 Only)	2-4
	2.2.3	NOVIR.OBJ	2-5
	2.2.4	VIRP.OBJ	2-5
	2.2.5	VIRNP.OBJ	2-5
	2.2.6	Converting a Program to Use VIRTUAL Arrays	2-6

CONTENTS

			Page
	2.3	SUBPROGRAM LINKAGE	2-9
	2.3.1	Subprogram Register Usage	2-11
	2.4	VECTORED ARRAYS	2-12
	2.5	PROGRAM SECTIONS	2-14
	2.5.1	Compiled Code PSECT Usage	2-14
	2.5.2	Common Block PSECT Usage	2-14
	2.5.3	OTS Library PSECT Usage	2-15
	2.5.4	Ordering of PSECTs in Executable Programs	2-16
	2.6	TRACEBACK FEATURE	2-17
	2.7	RUN-TIME MEMORY ORGANIZATION (RT-11 ONLY)	2-18
CHAPTER	3	FORTRAN IV SPECIFIC CHARACTERISTICS	3-1
	3.1	OPEN/CLOSE STATEMENT RESTRICTIONS	3-1
	3.1.1	Keyword Constraints	3-2
	3.2	SOURCE LINES	3-3
	3.3	VARIABLE NAMES	3-4
	3.4	INITIALIZATION OF COMMON VARIABLES	3-4
	3.5	CONTINUATION LINES	3-4
	3.6	STOP AND PAUSE STATEMENTS	3-4
	3.7	DEVICE/FILE DEFAULT ASSIGNMENTS	3-5
	3.8	MAXIMUM RECORD LENGTHS	3-6
	3.9	DIRECT ACCESS I/O	3-6
	3.9.1	DEFINE FILE Statement	3-6
	3.9.2	Creating Direct Access Files	3-6
	3.10	INPUT/OUTPUT FORMATS	3-6
	3.10.1	Formatted I/O	3-7
	3.10.2	Unformatted I/O	3-7
	3.10.3	Direct-Access I/O	3-8
	3.11	MIXED-MODE COMPARISONS	3-9
	3.12	FORTRAN BUFFERED I/O	3-9
CHAPTER	4	INCREASING FORTRAN IV PROGRAMMING EFFICIENCY	4-1
	4.1	FACTORS AFFECTING PROGRAM EFFICIENCY	4-1
	4.2	INCREASING COMPILATION EFFECTIVENESS	4-1
	4.3	PROGRAMMING TECHNIQUES	4-5
CHAPTER	5	CONCISE COMMAND LANGUAGE OPTION	5-1
	5.1	INTRODUCTION TO THE RSTS/E FORTRAN IV CCL OPTION	5-1
	5.2	COMMAND INTERFACE	5-1
	5.2.1	CCL Command Restrictions	5-2
	5.2.2	CCL Command Comparison	5-2
APPENDIX	A	FORTRAN DATA REPRESENTATION	A-1
	A.1	INTEGER FORMAT	A-1
	A.2	FLOATING-POINT FORMATS	A-1
	A.2.1	REAL Format (Two-Word Floating Point)	A-2
	A.2.2	DOUBLE PRECISION Format (Four-Word Floating Point)	A-2
	A.2.3	COMPLEX Format	A-2
	A.3	LOGICAL*1 FORMAT	A-3
	A.4	HOLLERITH FORMAT	A-3
	A.5	LOGICAL FORMAT	A-3
	A.6	RADIX-50 FORMAT	A-4

CONTENTS

			Page
APPENDIX	B	LIBRARY SUBROUTINES	B-1
	B.1	LIBRARY SUBROUTINE SUMMARY	B-1
	B.2	ASSIGN	B-2
	B.3	OPEN (RSTS/E ONLY)	B-4
	B.4	CLOSE	B-6
	B.5	DATE	B-6
	B.6	IDATE	B-7
	B.7	EXIT	B-7
	B.8	USEREX	B-7
	B.9	RANDU,RAN	B-8
	B.10	SETERR	B-8
	B.11	ERRTST	B-8
	B.12	ERRSNS	B-9
APPENDIX	C	FORTRAN IV ERROR DIAGNOSTICS	C-1
	C.1	COMPILER ERROR DIAGNOSTICS	C-1
	C.1.1	Errors Reported by the Initial Phase of the Compiler	C-2
	C.1.2	Errors Reported by Secondary Phases of the Compiler	C-4
	C.1.3	Warning Diagnostics	C-12
	C.1.4	Fatal Compiler Error Diagnostics	C-13
	C.2	OBJECT-TIME SYSTEM ERROR DIAGNOSTICS	C-14
APPENDIX	D	COMPATIBILITY WITH OTHER PDP-11 LANGUAGE PROCESSORS	D-1
	D.1	FORTRAN IV COMPATIBILITY WITH FTN V08.04	D-1
	D.1.1	Language Differences	D-1
	D.1.2	Implementation Differences	D-2
	D.2	DIFFERENCES BETWEEN FORTRAN IV-PLUS AND FORTRAN IV	D-3
	D.2.1	Language Differences	D-3
	D.2.2	Implementation Differences	D-3
	D.3	RSTS/E FORTRAN IV FILE COMPATIBILITY	D-4
	D.3.1	Sequential Stream ASCII Files	D-4
	D.3.2	Virtual Array Files	D-5
	D.3.3	BASIC-PLUS Record I/O Files	D-6
APPENDIX	E	THE FORTRAN IV SYSTEM SIMULATOR ($SIMRT)	E-1
	E.1	$SIMRT CAPABILITIES AND RELATIONSHIP TO RT-11	E-1
	E.2	$SIMRT TERMINAL HANDLING	E-1
	E.3	SYSLIB CALLS UNDER $SIMRT	E-2
	E.4	MODIFYING SIMRT	E-3
INDEX			Index-1

CONTENTS

Page

FIGURES

FIGURE	1-1	Steps in Compiling and Executing a FORTRAN IV Program	1-1
	1-2	A Sample Compilation Listing	1-13
	1-3	Finding the Base Address Section	1-32
	2-1	Array Vectoring	2-13
	2-2	The Traceback Feature	2-18
	2-3	RT-11 8K System Run-Time Memory Organization	2-19

TABLES

TABLE	1-1	Valid Options vs. Configuration	1-2
	1-2	Threaded Code Error Messages	1-3
	1-3	Device Specifications	1-5
	1-4	File Name Types (Extensions)	1-6
	1-5	File Protection Codes	1-6
	1-6	Compiler Options (Switches)	1-9
	1-7	Linker Options Available Under RT-11	1-18
	1-8	Linker Options (Switches) Under RSTS/E	1-21
	2-1	Comparison of Threaded and In-line Code for the Statement: I=J*K+REAL	2-3
	2-2	Return Value Convention for Function Subprograms	2-11
	2-3	Compiler Organization of Program Sections	2-14
	2-4	Organization of OTS Library Modules	2-15
	3-1	Keyword Constraints Under RT-11 and RSTS/E	3-2
	3-2	FORTRAN Logical Device Assignments	3-5
	A-1	ASCII/Radix-50 Equivalents	A-4

PREFACE

MANUAL OBJECTIVES

The RT-11, RSTS/E FORTRAN IV User's Guide is intended for use in developing new FORTRAN programs, and compiling and executing existing FORTRAN programs on RT-11 and RSTS/E systems. FORTRAN IV language elements are described in the PDP-11 FORTRAN Language Reference Manual.

INTENDED AUDIENCE

This manual should be used only after some knowledge of the FORTRAN language, as implemented on the PDP-11, has been acquired. The associated document that can be used for this purpose is the PDP-11 FORTRAN Language Reference Manual. The user should also be familiar with the operating system as described in either the RT-11 System User's Guide or the RSTS/E System User's Guide. The RSTS/E Documentation Directory and the RT-11 Documentation Directory contain additional information on the respective documentation sets.

STRUCTURE OF THIS DOCUMENT

This manual is organized as follows:

- Chapter 1, "Operating Procedures," contains the information needed to compile, link, and execute FORTRAN IV programs. It includes a new section about virtual jobs, and covers additional linker options.
- Chapter 2, "FORTRAN IV Operating Environment," describes how to use the facilities of the PDP-11 FORTRAN IV Object-Time System. It includes information about virtual and vectored arrays, program sections, and run-time error detection and memory organization.
- Chapter 3, "FORTRAN IV Specific Characteristics," discusses FORTRAN IV access methods and input/output, including information on logical device assignments and record structure.
- Chapter 4, "Increasing FORTRAN IV Programming Efficiency," covers programming considerations relevant to typical FORTRAN IV applications.
- Chapter 5, "Concise Command Language Option," describes how to invoke RSTS/E system programs using the Concise Command Language.
- Appendix A, "FORTRAN Data Representation," summarizes internal data representation.

- Appendix B, "Library Subroutines," describes user-accessible FORTRAN Library subroutines.

- Appendix C, "FORTRAN IV Error Diagnostics," describes compiler and Object-Time System diagnostic messages. It includes a number of new messages.

- Appendix D, "Compatibility with Other PDP-11 Language Processors," covers FORTRAN language and implementation differences.

ASSOCIATED DOCUMENTS

The following documents are relevant to RT-11 and RSTS/E FORTRAN IV programming:

- PDP-11 FORTRAN Language Reference Manual
- RSTS/E FORTRAN Utilities Manual
- RSTS/E System User's Guide
- RT-11 System User's Guide

DOCUMENTATION CONVENTIONS

All monitor and system program command lines are terminated by pressing the RETURN key. Since this is a nonprinting character, at certain places in the text the notation (RET) represents the RETURN key.

In examples, user responses shown in upper case characters indicate that you should type the characters exactly as shown.

In format descriptions, uppercase characters represent information that must be entered exactly as shown; lowercase characters represent variable information that must be supplied by the user.

Some special keyboard characters require that the CTRL (control) key be pressed simultaneously with a second character. These characters are denoted by ^ (up arrow) for example; ^Z (CTRL Z).

Ellipsis marks (...) indicate the omission of one or more words within a passage and show that the passage continues in the same vein.

CHAPTER 1

OPERATING PROCEDURES

1.1 USING THE FORTRAN IV SYSTEM

Figure 1-1 outlines the steps required to prepare a FORTRAN IV source program for execution under the RT-11 or RSTS/E executive: (1) compilation, (2) linking, and (3) execution.

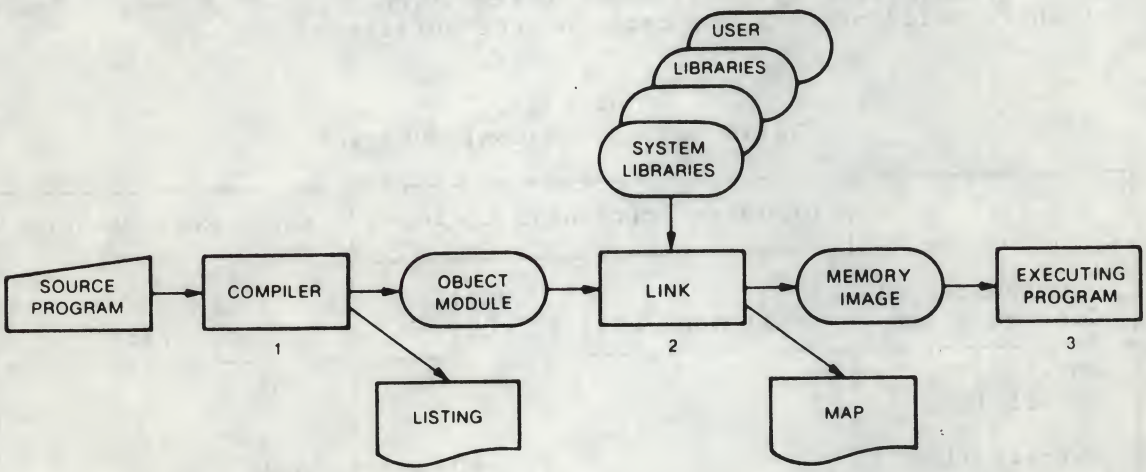


Figure 1-1 Steps in Compiling and Executing a FORTRAN IV Program

Step 1 in Figure 1-1 is initiated by running the FORTRAN IV Compiler, FORTRAN, accompanied by a command string that describes the input and output files, and desired options to be used by the compiler. The compiler generates an object file which must be linked by the Linker prior to execution.

Step 2 is initiated by running the Linker, LINK, accompanied by a similar command string. The Linker combines all program units and the necessary routines from the FORTRAN Library, and generates a memory image file.

Step 3 is initiated by the monitor RUN command.

1.1.1 Compiler-Generated Code

The FORTRAN IV compiler translates the symbolic (FORTRAN) program into an object program in binary form, resulting in machine instructions. If you find that procedures and instructions in a high level language like FORTRAN sometimes restrict your freedom to handle data as you would like, you can insert an assembly language routine as you find it desirable. This process is covered more fully in Section 2.3.

The FORTRAN IV compiler produces two types of object code for a program:

in-line (PDP-11 machine language)

threaded (linked OTS references)

See Section 2.2 for a further description of the object code produced. The default value assumed for the /CODE (or /I) option (see Section 1.1.1.1) is determined at installation time to satisfy the configuration in which the compiler is installed.

1.1.1.1 **Code Options** - In-line code is selected for RT-11 through the keyboard monitor command* options /CODE:EAE, /CODE:EIS, or /CODE:FIS; and for RSTS/E through options /I:EAE, /I:EIS, or /I:FIS. Throughout this guide, the RT-11 option is given first, followed by the RSTS/E equivalent in parentheses. Thus: /NOLINENUMBERS (/S). Take care to select the option suitable to the available hardware configuration. Some configurations will not support execution of in-line code. Table 1-1 shows valid options for certain configurations.

Table 1-1
Valid Options vs Configuration

Hardware Configuration	Hardware Arithmetic Options			Valid Code Options
	KE11-A,B	KE11-E	KE11-F KE11	/CODE: (/I:)
PDT-11/130, PDT-11/150	-	-	-	THR
LSI-11, 11/03	-	-	NO	THR
	-	-	YES	EIS, FIS or THR
11/04, 11/05, 11/10, 11/15, 11/20	NO	-	-	THR
	YES	-	-	EAE or THR
11/35, 11/40	NO	NO	NO	THR
	YES	NO	NO	EAE or THR
	NO	YES	NO	EIS or THR
	NO	YES	YES	EIS, FIS or THR
11/23, 11/34, 11/44, 11/45, 11/50, 11/55, 11/60, 11/70	-	-	-	EIS or THR

1.1.1.2 **Code Selection and Error Messages** - When the compiler produces in-line code (/CODE: [/I:] followed by EAE or EIS or FIS) the object program executes at greater speed and generally uses less physical memory. In-line code achieves this optimization, in part, by omitting instructions to detect or report certain error conditions. However, you can generate code for error checking by including the /CODE:THR (/I:THR) option in the compiler command line. Table 1-2 demonstrates the diagnostic benefits of the threaded code option.

Table 1-2
Threaded Code Error Messages

Error	Result	
	in-line code	threaded code
1. The result of an integer multiply operation can not be expressed as a one-word integer.	No diagnostic message is produced. Execution continues and the result of the operation is truncated to 16 bits.	A fatal error occurs and the diagnostic message: "?Err 1 Integer Overflow" is produced.
2. A divide by zero occurs during integer arithmetic.	No diagnostic message is produced and the result of the operation is undefined.	A fatal error occurs and the diagnostic message "?Err 2 Integer zero divide" is produced.
3. The value of the arithmetic expression of a computed GOTO is less than one or greater than the number of labels in the list.	No diagnostic message is produced. Execution continues at the next executable statement.	The warning diagnostic "?Err 4 Computed GOTO out of range" is produced. Execution resumes at the next executable statement.

1.1.2 File Name Specifications

The FORTRAN and LINK commands, respectively, pass file name specifications to the FORTRAN IV compiler and Linker. The designator .typ (type) is used for RT-11. The RSTS/E equivalent is .ext (extension). See Table 1-3 for device specifications, and Section 1.2.2 for compiler options.

Each file name specification (filespec) has the form:

RT-11	RSTS/E
dev:filnam.typ	dev:[p,pn]filnam.ext<prot>/sw

where

dev: is an optional two to three character name (up to six characters for RSTS) specifying a legal device code as shown in Table 1-3 for a logical device name. If the device code is omitted, the default storage (DK:) is used for RT-11 and the public structure (SY:) is used for RSTS/E.

filnam is any one to six character alphanumeric file name.

.typ (RT-11)
.ext (RSTS/E) is any zero to three character alphanumeric extension.

If one is not specified, the FORTRAN IV Compiler supplies, by default, certain extensions as shown in Table 1-4.

The following apply to RSTS/E only:

[p,pn] is a RSTS/E project (p), programmer number (pn), which is used to identify the account under which the file is stored.

<prot> is a RSTS/E protection code restricting access to a file. The degree of restriction is determined by a code or combination of codes, as shown in Table 1-5. Protection codes have effect only on output files.

/sw is a RSTS/E option consisting of either or both the following RSTS/E file specification switches. (Refer to the RSTS/E Programming Manual for further information.)

/CL:n set clustersize of (output) file to n.

/MO:n use mode n when opening the file.

The protection code is a string of one to three decimal digits enclosed by angle brackets <>. The protection code determines the file's degree of protection on two levels: the actions - reading, writing, and deleting - against which it is protected, and the user or class of users against whom it is protected. There are three such user classes, which the system recognizes by project-programmer numbers:

1. The individual user (owner), who is recognized by his programmer number: [200,25].

2. The user's project group, which is recognized by the user's project number: [200,25],[200,57],[200,70].

3. All other users on the system, who are recognized by the existence of valid project-programmer numbers: [225,60],[250,35],[254,10].

Typically, a file's total protection code is the sum of the desired combination of individual codes. A data file with protection <60> - the usual system default - is protected against reading, writing, and deleting by all users except its owner: <60>=4+8+16+32. For detailed information, see the RSTS/E System User's Guide.

Table 1-3
Device Specifications

Device	RT-11	RSTS/E
Card reader	CR:	CR:
TA11 cassette (n=0 or 1)	CTn:	-
Default storage	DK:	SY:
RK05 disk (n=0 to 7)		
RK06 or RK07 disk (n=0 to 7)	DMn:	DMn:
RP02 or RP03 disk	DPn:(n=0 to 1)	DPn:(n=0 to 7)
RL01 or RL02 disk	DLn:(n=0 to 4)	DLn:
RM02 or RM03 disk (n=0 to 7)	-	DRn:
RP04, RP05 or RP06 disk (n=0 to 7)	-	DBn:
RS03 or RS04 disk (n=0 to 7)	DSn:	DSn:
TC11 DECtape	DTn:(n=0 to 7)	DTn:(n=0 to 7)
TU58 DECtape II	DDn:(n=0 to 4)	DDn:(n=0 to 7)
PDT-11/130 DECtape II(n=0 to 1)	PDn:	-
RX01 floppy disk	DXn:(n=0 to 3)	DXn:(0 to 7)
PDT-11/150 floppy disk	PDn:	
Serial line printer	LS:	
RX02 floppy disk	DYn:(n=0 to 3)	DYn:(n=0 to 7)
Line printer	LP:	LPn:(n=0 to 7)
TU16, TE16, TU45, or TU77 magtape (n=0 to 7)	MMn:	MMn:
TU10, TE10, or TS03 magtape (n=0 to 7)	MTn:	MTn:
TS11 magtape (n=0 to 3)	MSn:	MSn:
High speed paper tape punch	PC:	PP:
High speed paper tape reader	PC:	PR:
RF11 fixed-head disk drive	RF:	DF0:
System device	SY:	SY:
Specified unit from which the system was bootstrapped	SYn:	SY0:
Current user terminal	TT:	KB:
Auxiliary terminal	-	KBn:

For more information on device specifications, refer to the RT-11 System User's Guide and the RSTS/E System User's Guide.

Table 1-4
File Name Types (Extensions)

File	Default Type (Extension) on Output File
Source file	.FOR
Object file	.OBJ
Listing file	.LST
Load Map file	.MAP
Save Image file	.SAV
Absolute Binary file	.LDA (/LDA)
Relocatable Image file	.REL (/R) Foreground (RT-11 only)

Table 1-5
File Protection Codes

Code	Meaning
1	Read protection against owner
2	Write protection against owner
4	Read protection against owner's project group
8	Write protection against owner's project group
16	Read protection against all others who do not have owner's project number
32	Write protection against all others who do not have owner's project number
64	Executable program: can be run only
	Individual codes added to the compiled protection <64> have meanings different from those of the data file protection codes above. These compiled codes follow:
1	Execute protection against owner
2	Read and write protection against owner
4	Execute protection against owner's project group
8	Read and write protection against owner's project group
16	Execute protection against all others who do not have owner's project number
32	Read and write protection against all others who do not have owner's project number
128	Program with temporary privileges (normally occurs only when file's protection includes <64>)

1.1.3 Locating a File

The FORTRAN IV compiler locates a file by searching the specified device for the file name with the specified file type (RT-11), or extension (RSTS/E). The compiler searches default storage (public structure on RSTS/E) when the device is not specified and assumes .FOR when a file type or extension is not specified.

Under the RSTS/E operating system, the computer seeks the specified account [project, programmer] number. When the account number is not specified, the compiler searches the current user's directory and proceeds to the system library [1,2] if the file is not in the user's directory. The protection code identifies the read and write access to be granted the user of the located file. If a protection code is not specified on output, the system default code (usually 60) is used.

If the file cannot be located or is protected against the user, the compiler prints the following message:

```
?FORTRAN-F-FILE NOT FOUND
```

A similar form of this message appears if a file name specification given to a utility program (such as, MACRO, LINK,) references a file that cannot be found or is protected against the user.

1.2 RUNNING THE FORTRAN IV COMPILER

The FORTRAN IV Compiler accepts a command string of the form:

```
output = input/option (/sw)
```

where

output	is the output file name specification(s).
input	is the input file name specification(s).
/option (RT-11) /sw (RSTS/E)	is one or more options used to request certain functions from the FORTRAN IV Compiler. Options are covered in Section 1.2.2. Options may be appended to any file specification in the command string.

Note that imbedded blanks are not permitted in command string specifications.

1.2.1 Under RT-11 (with Keyboard Monitor Commands)

To compile a FORTRAN program, either of these commands is given:

```
FORTRAN[/option...] filespec[/option...][...filespec[/option...]]
```

or

```
FORTRAN[/option...]
FILES?  filespec[/option...][...filespec[/option...]]
```

(where filespec represents a source file to be compiled).

When more than one file is listed for a single compilation, separate the files by plus (+) signs. FORTRAN will create an output file with the same name as the first input file and give it a .OBJ file type. However, if you separate the input files by commas, FORTRAN will produce an .OBJ file for each input file listed.

For example:

```
FORTRAN/LIST/SHOW:ALL/NOLINENUMBERS TEST1+TEST2
```

compiles TEST1.FOR and TEST2.FOR together and produces TEST1.OBJ. A listing of the compilation complete with source, storage map and code listings will be sent to the line printer device, LP:. Generation of Internal Sequence Numbers (ISNs) in the object program will be disabled.

1.2.1.1 **Under RSTS/E** - To compile a FORTRAN program, the command is given:

```
RUN $FORTRAN
*
-
```

The FORTRAN IV Compiler then prints an asterisk (*) to indicate that it is ready to accept a command string.

The FORTRAN IV Compiler can produce two output files: an object file and a listing file. As many as six FORTRAN IV source language files are permitted as input files. If multiple input files are given, they are considered to be logically concatenated. However, source lines must not be broken over file boundaries.

An input file that resides on a random access device can contain more than one program unit. The object code for all program units to be concatenated at LINK time goes to the resulting single object file.

A sample FORTRAN IV Compiler command sequence is shown below:

```
RUN $FORTRAN
*OBJECT,LIST=FILE1
```

This command string directs the compiler to take the source file FILE1.FOR from the current account on the public structure, and output the files LIST.LST and OBJECT.OBJ to the current account on the public structure.

Either of the compiler output files can be eliminated by omitting its file specification from the command string. For example:

```
RUN $FORTRAN
*FILE1=FILE1
```

produces FILE1.OBJ on the default device but no listing file, while

```
*,LP:=FILE1
```

produces a listing on the line printer, but no object module output.

1.2.2 Compiler Options (Switches)

The FORTRAN IV Compiler command strings utilize specified options (switches) of either octal or decimal values on the input and output file specifications. Any option of the form /S:n causes n to be interpreted as an octal value (as long as n contains only the digit 0-7); whereas /S:n. causes n to be interpreted as a decimal value. RT-11 and RSTS/E FORTRAN IV Compiler options (switches) are described in Table 1-6.

Table 1-6
Compiler Options (Switches)*
(defaults are determined at installation time)

RT-11 Option	RSTS/E Switch	Explanation
/ALLOCATE:n	-	Used after the /OBJECT or /LIST option to guarantee space for a maximum file size of n blocks.
/CODE:xxx	/I:xxx	Selects type of object code to be generated. Defaults to value selected at installation. The valid values are: EAE (selects EAE hardware) EIS (selects EIS hardware) FIS (selects EIS and FIS hardware) THR (selects threaded code)
/DIAGNOSE	/B	Enables expanded listings of compiler internal diagnostic information.
/EXTEND	/E	Allows source line input from columns 73-80.
/HEADER	/O	Prints an "Options-In-Effect" section prefacing the listing.
/I4	/T	Defaults to two word-integers (I*4) (normally defaults to one-word integers (I*2)).
/LINENUMBERS	-	Indicates internal sequence numbers are to be included in the executable program for routine diagnostics.
/LIST[:filespec]	-	Generates a listing. A file name can be optionally specified.
/NOLINENUMBERS	/S	Suppresses generation of internal sequence numbers.
/NOOBJECT	-	Does not generate object files.

* RT-11 users may use the RSTS/E switch when the compiler is invoked with the RUN command.

(continued on next page)

Table 1-6 (Cont.)
Compiler Options (Switches)*
(defaults are determined at installation time)

RT-11 Option	RSTS/E Switch	Explanation
*	/Q	Inhibits printing names of program units (from program, FUNCTION, SUBROUTINE, and BLOCK DATA statements) as each program unit is compiled. Note that .MAIN. refers to the main program and .DATA. refers to an unnamed BLOCK DATA.
/NOSWAP	/U	Disables USR swapping at run time.
/NOVECTORS	/V	Suppresses array vectoring of multidimensional arrays.
/OBJECT[:filespec]	-	Produces an object file (default). The destination for the object file can be optionally specified.
/ONDEBUG	/D	Compile lines with a "D" in column one (for debugging purposes).
*	/Z	Causes pure code and pure data sections to take RO (read-only) attribute.
/RECORD:n	/R:n	Specifies the maximum record length (in bytes) on run time I/O (4<n<4095).
/SHOW[:n]	/L[:n]	Specifies the listing options. The argument n is encoded as follows: 0 or null - list diagnostics only 1 or SRC - list source program and diagnostics only 2 or MAP - list storage map and diagnostics only 4 or COD - list generated code and diagnostics only Any combination of the above list options may be specified by summing the numeric argument values for the desired list options. For example: 7 or ALL requests a source listing, a storage map, and a generated code listing. If this option is omitted, the default option is /SHOW:3, (/L:3) source and storage map.

* RT-11 users may use the RSTS/E switch when the compiler is invoked with the RUN command.

(continued on next page)

Table 1-6 (Cont.)
Compiler Options (Switches)*
(defaults are determined at installation time)

RT-11 Option	RSTS/E Switch	Explanation
/STATISTICS	/A	Prints compilation statistics.
/SWAP	-	Allows the USR to swap over the FORTRAN program (default).
/UNITS:n	/N:n	Allows a maximum of n simultaneously open I/O channels at run time (1<n<15).
/VECTORS	-	Uses tables to access multidimensional arrays (default).
/WARNINGS	/W	Enables compiler warning diagnostics; used in conjunction with /SHOW (/L) option.
*	/X:xxx	Indicates cross-compilation for the target environment specified. Compiler diagnostic messages will be generated as if compilation had occurred under the foreign environment. Values are: RT (selects RT-11) RST (selects RSTS/E) RSX (selects RSX-11)

* RT-11 users may use the RSTS/E switch when the compiler is invoked with the RUN command.

1.2.3 Listing Formats

You can direct the compiler to furnish any combination of five optional sections in the compilation listing. Use the options (switches) described in Section 1.2.2 to call for the list of options in effect, the generated code and the compiler statistics. The source program and the storage map are included by default. Figure 1-2 describes each section and gives an example of the information included.

FORTRAN IV listings of generated code list the first instruction starting at location 6 of the procedure. This section explains the two instructions generated that are not listed.

The first two lines of code are not generated in the FORTRAN IV listings for either subroutines or the main segment.

In the case of the main program unit, the two lines generated are:

```
JSR     R4,$$OTI                    (IN-LINE CODE)
.WORD   NAMPTR
```

or

```
JSR     R4,$OTI                     (THREADED CODE)
.WORD   NAMPTR
```

The location that NAMPTR contains is the address of the two-word segment name in RAD50, which is the name of the main program. If no main-program name is specified by means of a PROGRAM statement, the default main-program name, .MAIN., is used.

In a subprogram unit, the two lines generated are:

```
JSR     R4,$OTIS
.WORD   NAMPTR
```

The location that NAMPTR contains is the address of the two-word segment name in RAD50, which is the name of the routine.

```
FORTRAN IV      V02.5      Thu 01-May-80 00:40:58

,EX2=EX2/L:ALL/I:THR/O/A

OPTIONS IN EFFECT:

                    SOURCE
                    MAP
                    CODE
                    LEAPYEAR
                  NOREADONLY
                    LRECL=0136
                    STAT
                    ISNS
                  NOCOL80
                    USRSWAP
                  NODIAGNOSE
                  NOINTEGER*4
                    NLCHN=06
                  NODEBUG
                    VECTOR
                  NOWARN
                    CODE:THR
                    LOG
```

```
FORTRAN IV      V02.5      Thu 01-May-80 00:40:58

0001          INTEGER INT
0002          REAL REAL
0003          COMPLEX IMAG
0004          DOUBLE PRECISION DBLE
0005          DATA INT/100/
0006          REAL = INT/2 + 5.
0007          DBLE = REAL/2. + 3.1415926535D0
0008          IMAG = CMPLX(REAL, 3.21 )
0009          WRITE (5,10) IMAG
0010    10    FORMAT(1X,2F8.5)
0011          STOP
0012          END
```

```
FORTRAN IV      Storage Map for Program Unit .MAIN.

Local Variables, .PSECT $DATA, Size = 000030 (   12. words)

Name   Type  Offset     Name   Type  Offset     Name   Type  Offset
DBLE   R*8   000020     IMAG   C*8   000010     INT    I*2   000002
REAL   R*4   000004

Subroutines, Functions, Statement and Processor-Defined Functions:

Name   Type   Name   Type   Name   Type   Name   Type   Name   Type
CMPLX   C*8
```

Figure 1-2 A Sample Compilation Listing

```
FORTRAN IV        Generated Code for Program Unit .MAIN.

Statement #0006
000006   LSN$      #000006
000012   MOI$MS    $DATA+#000002
000016   DII$IS    #000002
000022   CFI$
000024   ADF$IS    #040640
000030   MOF$SM    $DATA+#000004

Statement #0007
000034   ISN$
000036   MOF$MS    $DATA+#000004
000042   DIF$IS    #040400
000046   CDF$
000050   ADD$MS    $DATAP+#000020
000054   MOD$SM    $DATA+#000020

Statement #0008
000060   ISN$
000062   REL$      $DATAP+#000030
000066   REL$      $DATA+#000004
000072   CAL$      #000002 CMPLX+#000000
000100   MOD$RM    $DATA+#000010

Statement #0009
000104   ISN$
000106   REL$      $DATAP+#000016
000112   REL$      $DATAP+#000010
000116   IFW$
000120   REL$      $DATA+#000010
000124   TVC$
000126   EOL$

Statement #0011
000130   LSN$      #000013
000134   STP$


Compilation Statistics:

Symbol table size: 00091 words
Internal form size: 00039 words
Free dynamic memory: 19983 words

Compilation time: 00:00:01
```

Figure 1-2 A Sample Compilation Listing (Cont.)

1.2.3.1 **Options Listing** - Use the options-in-effect list as a quick reference to the status of each possible compiler option. Those preceded by 'NO' are not in effect. The maximum number of logical units that can be concurrently open (NLCHN) and the maximum record length (LRECL) are given as the default values or the values specified by the /UNITS:n (/N:n) and /RECORD:n (/R:n) options respectively. The day of the week, date, and time of compilation and a copy of the compiler command string for identification purposes are also furnished.

1.2.3.2 **Source Listing** - This section lists the source program as it appeared in the input file. The compiler adds internal sequence numbers for easier reference. Note that internal sequence numbers are not always incremented by 1. For example, the statement following a logical IF has an internal sequence number two greater than that of the IF, because the compiler assigns one for the comparison and one for the associated statement.

1.2.3.3 **Storage Map Listing** - This section lists all symbolic names referenced by the program unit. Local variables are allocated in the $DATA psect. The addresses of parameter variables and arrays are placed in the $DATA psect at subroutine invocation and are denoted by "@" preceding the offset or section name. The listing includes the symbolic name, data type, usage, psect, and offset. In the case of COMMON blocks, VIRTUAL arrays, and array names, the listing includes the defined size in bytes (octal) and words (decimal) as well as the dimensions.

NOTE

Blank COMMON is described as COMMON BLOCK / / in the storage map, but is located on a LINK map as a PSECT named .$$$$.

1.2.3.4 **Generated Code Listing** - This section contains:

- a symbolic representation of the object code generated by the compiler (see Section 2.2).
- includes a location offset into psect $CODE
- the symbolic Object Time System (OTS) routine name
- routine arguments of threaded code plus the equivalent assembler code for the in-line code. When the /DIAGNOSE (/B) compiler option is used, the right-hand coulmn of the in-line generated code listing shows those registers available for use following an operation. The code generated for each statement provides easy cross-reference by showing the same internal sequence number (ISN) as was specified in the source program listing.

1.2.3.5 **Compilation Statistics** - This section provides a report on memory usage during the compilation process and elapsed wall-clock time for the compilation.

1.2.4 Compiler Memory Requirements

RT-11 and RSTS/E differ in the amount of memory available to each for compilation, and the options available for obtaining additional space. Under RT-11, device handlers and the symbol table require a portion of memory during compilation. When more memory is required after minimizing the number of different physical devices and variable names specified, you can segment the program into program units small enough to compile in the available space.

Under RSTS/E, you acquire additional space by switching from a nonprivileged to a privileged account, increasing the system swap maximum, or segmenting the program. These options and the amount of memory available are treated in greater detail in the following subsections.

1.2.4.1 **Compiler Memory Requirements Under RT-11** - During compilation, the following must reside in main memory: the RT-11 Resident Monitor (RMON), the compiler root segment, one overlay region, the stack, and the required device handlers (other than the handler for the system device, which is included in the RMON). Note that FORTRAN will dynamically load device handlers required for compilation. The remaining memory provides for the symbol table and the internal representation of the program. In a machine with 8K words of memory, this allows the compilation of a program unit as large as several hundred statements. However, if the compiler runs out of memory during compilation, an error message is delivered to the user's terminal; see Section C.1. The program must be divided into two or more program units, each small enough to compile in the available memory.

Since device handlers and the symbol table must be resident in memory during compilation, minimize the number of different physical devices specified in the command string and reduce the number of variable names to increase the amount of memory available for object code generation.

1.2.4.2 **Compiler Memory Requirements Under RSTS/E** - The RSTS/E FORTRAN IV compiler acquires the maximum free memory (up to 28K words) allocated to the current user. The private memory maximum for the user's account will never exceed the SWAP MAX currently set by the system manager, who may also restrict the dynamic memory requested by FORTRAN IV for every user.

If the compiler runs out of free space during a compilation, an error message is delivered to the user's terminal; see Section C.1. In this case, you can take several actions to accommodate the compilation:

1. If the compilation was attempted by a nonprivileged user whose private swap maximum is smaller than the system swap maximum, the compilation may proceed, but under a privileged account. This may allow the compiler to acquire a larger memory area. Or, you can request the system manager to increase the private swap maximum.

2. If a compilation terminates with insufficient space under a privileged account, do either of the following:

   - Ask the system manager to increase the FORTRAN compilation size limit to accommodate large compilations, or
   - Segment the program unit into two or more smaller program units, and reduce the number of variables, arrays, and constants to save compiler symbol table space.

1.3 LINKING PROCEDURES

When SYSLIB is created under RT-11 or RSTS/E to include FORLIB, the /LINKLIBRARY:FORLIB (/F) option is redundant, because FORLIB is part of SYSLIB and is not called separately.

1.3.1 Linking Under RT-11

The RT-11 linker, LINK, combines one or more user-written program units with selected routines from any user libraries and the default FORTRAN IV OTS Library to form the default system subroutine library, SYSLIB. LINK generates a single runnable memory image file and an optional load map from the one or more object files created by the MACRO assembler or the FORTRAN IV compiler.

The default types for the executable file are .SAV for a background or mapped environment program, and .REL for a foreground program. The default output device is DK:.

The default name of the .SAV or .REL file is that of the first concatenated input object file specified. When FORLIB resides in SYSLIB, the required elements of the FORTRAN library will be linked automatically since any undefined global references are correlated and resolved through SYSLIB.

The LINK command adheres to the following syntax:

```
LINK[/option...] filespec[/option...][,...filespec[/option...]]
```

or

```
LINK[/option...]
FILE? filespec[/option...][,...filespec[/option...]]
```

where "filespec" represents the file to be linked and "options" are those described in Table 1-7.

Table 1-7
Linker Options Available Under RT-11

Option	Explanation
/ALLOCATE:n	Guarantees space for a maximum file of n blocks.
/ALPHABETIZE	Lists program's global symbols alphabetically in the load map.
/BITMAP	Creates a memory usage bitmap (default setting).
/BOTTOM:n	Specifies a bottom address for a background program.
/BOUNDARY:value	Starts a specific program section on a particular address boundary. Argument value, must be a power of 2. Prompts you: Boundary section? Enter name of section, then RET.
/DEBUG[:filespec]	Links ODT to the linked program.
/EXECUTE[:filespec]	Designates the executable file.
/EXTEND:n	Extends a program section to octal value n. Prompt: Extend section?
/FILL:n	Initializes unused locations in the load module to n (an octal value).
/FOREGROUND [:stacksize]	Generates a .REL file for a foreground link.
/INCLUDE	Allows subsequent entry at the keyboard of global symbols to be taken from any library and included in the linking process. When the /INCLUDE option is typed, the linker prints: Library search? Reply with the list of global symbols to be included in the load module. Press the carriage return key RET to enter each symbol in the list.
/LDA	Produces executable file in LDA format for use with the Absolute Loader.
/LIBRARY	Same as /LINKLIBRARY. (Included for compatibility with other systems.)

(continued on next page)

Table 1-7 (Cont.)
Linker Options Available Under RT-11

Option	Explanation
/LINKLIBRARY:filespec	This option is ignored unless a file specification is typed. The file specification is included as an object module library in the linking operation.
/MAP[:filespec]	Produces a link map on the listing device LP: or in the file specified.
/NOEXECUTE	Does not create a .SAV file.
/PROMPT	Causes the LINKer and LIBRarian to prompt for CSI formatted commands. The LINKer/LIBRarian treat the command strings as continuation lines until a // is seen. /PROMPT is equivalent to // mode of continuation. Use this option to specify overlays, for example: .LINK/PROMPT ROOT *OVR1/O:1 *OVR2/O:1 *OVR3/O:2 *OVR4/O:2// This creates two overlay regions with two segments each.
/ROUND:n	Rounds up a section so that the root is a whole number multiple of n (a power of 2). Prompt: Round section:
/RUN	For background jobs only, executes the resulting SAV file.
/SLOWLY	Allows largest memory area for symbol table.
/STACK[:n]	Modifies the stack address (default is loc. 42). Give an octal value (:nnnnnn) or else system prompts for a global symbol: Stack symbol?
/SYMBOLTABLE[:filespec]	Creates a file containing symbol definitions for all global symbols. Enter the symbol table file specification as the third output specification in the LINK command.
/TOP:value	Specifies the highest address to be used by the relocatable code. The argument value represents an unsigned, even octal number.

(continued on next page)

Table 1-7 (Cont.)
Linker Options Available Under RT-11

Option	Explanation
/TRANSFER[:n]	Prompts for a global symbol to be used as the starting address of the program. The user can specify a starting address (represented by n).
/WIDE	Sets the number of columns for the width of the link map to 6. The default width is normally 3 for an 80 column wide listing.
/XM	Enables special .SETTOP features in the XM monitor. This option allows a virtual job to map a scratch region in extended memory with the .SETTOP programmed request. See the RT-11 Programmer's Reference Manual for further information on these special .SETTOP features.

Examples of linker options under RT-11 are:

1) `LINK A,B,C` — Links A.OBJ, B.OBJ, and C.OBJ on DK: and creates A.SAV on DK:

2) `LINK/MAP A` — Links A.OBJ and creates A.SAV on DK: and a map on LP:

3) `LINK/MAP:RK1:/EXE:RK0: A,B,C`
Links A.OBJ, B.OBJ, and C.OBJ. The map A.MAP goes to RK1: and the executable file A.SAV goes to RK0:.

4) `LINK/MAP/EXE:FOO B,C,D,E,LIB/LIB`
Links B.OBJ, C.OBJ, D.OBJ, E.OBJ, and the Library LIB.OBJ to create FOO.SAV on DK: and a map on LP:.

1.3.2 Linking Under RSTS/E

The LINK command under RSTS/E has the form:

```
RUN $LINK
* command string
```

The command string has the following format:

```
dev:binout,dev:mapout=dev:obj1,dev:obj2,.../s1/s2/s3
```

where

dev: is a random access device for the save image output file (binout), and any appropriate device in all other instances. If dev: is not specified, the default device is assumed. If the output is to be LDA format (that is, the /L switch was used), the output file need not be on a random access device.

binout is the name to be assigned to the linker's save image, or LDA format output file. This file is optional; if not specified, no binary output is produced. (Save image is the assumed output format unless the /L switch is used.)

mapout is the optional load map file.

objl,... are files of one or more object modules to be input to the linker (these may be library files).

/s1/s2/s3 are optional switches, as explained in Table 1-8.

An example of the LINK command format as used with FORTRAN IV follows:

```
RUN $LINK
*LOAD,MAP=MAIN,SUB1,SUB2/F
```

This command string requests LINK to combine the object module MAIN.OBJ with the object modules SUB1.OBJ and SUB2.OBJ into the single save image file LOAD.SAV. A load map file MAP.MAP is also produced. All files are on the public structure.

The switch, /F, specifies that the default FORTRAN Library in the system library account [1,2] on the public structure, SY:FORLIB.OBJ, is to be searched for any routines that are not found in the other object modules. These include any library functions, system subroutines, or object-time system routines. Note that the switch alone, without the explicit file specification, causes the default FORTRAN Library to be searched. This switch should be included if any of the object modules specified in the command string were created by the FORTRAN Compiler. This switch can be omitted, however, if the FORTRAN Library file specification, SY:$FORLIB, is explicitly included in the command string or if FORLIB has been installed under the name SYSLIB, as illustrated in the following example.

```
RUN $LINK
*LOAD,MAP=MAIN
```

The optional load map file specification, if included, requests the linker to output a list of module names, common blocks, and global symbols, together with their absolute memory address assignments.

See the RSTS/E FORTRAN IV Utilities Manual for a more detailed description of LINK. Refer to Chapter 5 of this guide for an alternative procedure for invoking LINK.

Table 1-8
Linker Options (Switches) Under RSTS/E

Option (Switch) Name	Command Line	Meaning
/A	First	Alphabetizes the entries in the load map.
/B:n	First	Bottom address of program is indicated as n. The bottom address determines the amount of stack (SP) space available to the program being linked. The default bottom is 1000 (octal), which provides approximately 80 words of stack. This can be increased by specifying the /B switch with an argument greater than 1000 (octal).

(continued on next page)

Table 1-8 (Cont.)
Linker Options (Switches) Under RSTS/E

Option (Switch) Name	Command Line	Meaning
/C	Any	Continues input specification on another command line. Used also with /O.
/E:n	First	Allows a specified program section to be extended to the value given. When the /E switch is specified, the linker prints: EXTEND SECTION? Reply with the name of the program section, whose length then becomes greater than or equal to the value given. It will be "greater than" when the object code requires a space larger than the value specified.
/F	First	Instructs the linker to use the default FORTRAN library, FORLIB.OBJ, to resolve any undefined global references. Note that this option is not specified in the command line when FORLIB has been incorporated into SYSLIB.
/H:n	First	Specifies the top (highest) address to be used by the relocatable code in the load module. The high value must be specified or the error message /H NO VALUE will be returned. The high value must be an unsigned even octal number. If the value is odd, /H ODD VALUE error is returned. If the value is not large enough for the relocatable code, /H VALUE TOO LOW error message is returned. Use care with the /H switch because most RT-11 programs use the free memory above the relocatable code as a dynamic working area for I/O buffers, device handlers, symbol tables, etc. The size of this area varies with different memory configurations as programs linked to a high address may not run in a system with less physical memory. /R, /B, and /H are mutually exclusive and give the error /x-BAD SWITCH. /H is the counterpart to /B.
/I	First	Includes in the core image (see Section 1.3.3) the library object modules that declare the specified global symbols.
/K:n	First	Intended for RSTS/E; not normally used for RT-11. Puts the specified value in word 56 of the image file block 0. This value states that the program requires nK words of memory. Range for the required value is 1 through 28.
/L	First	Produces an output file in LDA format.

(continued on next page)

Table 1-8 (Cont.)
Linker Options (Switches) Under RSTS/E

Option (Switch) Name	Command Line	Meaning
/M or /M:n	First	Specifies the stack address at the terminal keyboard or via n.
/O:n	Any but the first	Indicates that the program has an overlay structure: n specifies the overlay region to which the module is assigned.
/R	First	(RT-11 only). Produces an output file in relocatable image format for execution as a foreground job.
/S	First	Allows the maximum amount of space in memory to be available for the linker's symbol table. (This switch should only be used when a particular link stream causes a symbol table overflow.)
/T or /T:n	First	Specifies the transfer address at terminal keyboard or as the octal value for the switch.
/U:n	First	Prompts the user with "ROUND SECTION": The user replies with the name of the program section to be rounded up, that must be in the root segment. The value given must be a power of 2. The specified section will be rounded up to a size that is a whole number multiple of n. An example would be to make the first overlay start on a block boundary (/U:1000), so that the root section and the first overlay region can be read in with only one read. If the specified section is not found, the error message will be "ROUND SECTION NOT FOUND".
/W:n	First	Specifies the width of the map to be produced. The value is the number of ENTRY ADDR combinations to print across the page. If no /W is given, the default is 3 (normal for 80 column paper). If only /W is given, n defaults to 6, which is ideal for a 132 column page. Useful range is 1 through 7.
/X	First	Intended for RSTS/E; not normally used for RT-11. Meaning: do not output (Xmit) the bitmap if code below 400. Locations 360-377 in block 0 of the load module are used for the bitmap. The linker normally stores the program usage bits in these eight words. Each bit represents 256-word block of memory. This information is used by the R, RUN, and GET commands when loading the program. Therefore care should be exercised in using this switch.

(continued on next page)

Table 1-8 (Cont.)
Linker Options (Switches) Under RSTS/E

Option (Switch) Name	Command Line	Meaning
//	First and last	This method provides an alternative to the /C (Continue) switch, which must be given on every line except the last. The // switch allows additional lines of command string input. // is typed on the first command line and the linker will continue to request input until the next occurrence of //. The second occurrence of // terminates specification of command string input. The second occurrence may be on the last command line with an object file name or on a command line by itself. CAUTION: The use of /C and // cannot be mixed in a link command string input sequence. Example: .R LINK *LINK,LP:=LINK0/B:500/W// *LNKOV1/0:1 . . . *LNKOV8/0:1//

1.4 LIBRARY USAGE

You can create a library of commonly used assembly language and FORTRAN functions and subroutines through the system program, LIBR, which provides for library creation and modification. The librarian chapter of the RT-11 System User's Guide or the RSTS/E FORTRAN IV Utilities Manual describes the LIBR program in detail.

Include a library file in the LINK command string simply by adding the file specification to the input file list. LINK recognizes the file as a library file and links only the required routines. The LINK command string;

```
*LOAD=MAIN,LIB1/F
```

requests LINK to combine MAIN.OBJ with any required functions or subroutines contained in LIB1.OBJ. The default FORTRAN system library, FORLIB.OBJ ($FORLIB.OBJ on RSTS/E), is then searched for any other required routines. /F is not specified if FORLIB has been incorporated into SYSLIB. At this point, any unresolved GLOBALS are resolved through SYSLIB. The entire memory image is output to the file LOAD.SAV.

If the /F option or switch is used, all user-created libraries are searched before the default FORTRAN system library, FORLIB.OBJ ($FORLIB.OBJ on RSTS/E). Consult the linker chapter of the RT-11 System User's Guide for a detailed description of multilibrary global resolution.

If the linker fails for lack of symbol table space, use the /S linker option in your next attempt. This could slow the linking process, but it allows the maximum possible symbol table space.

To maintain the integrity of the DEC-distributed FORTRAN Library, create a user library rather than modifying or adding to the FORTRAN Library (FORLIB) or to the System Library (SYSLIB).

1.4.1 Overlay Usage

Use the overlay feature of the linker to segment the memory image so that the entire program is not memory-resident at one time. This allows the execution of a program too large for the available memory.

An overlay structure consists of a root segment and one or more overlay regions. The root segment contains the FORTRAN IV main program, COMMON, subroutines, function subprograms, and any .PSECT that has the GBL attribute and is referenced from more than one segment. An overlay region is an area of memory allocated for two or more overlay segments, only one of which can be resident at one time. An overlay segment consists of one or more subroutines or function subprograms.

When a call is made at run time to a routine in an overlay segment, the overlay handler verifies that the segment is resident in its overlay region. If the segment is in memory, control passes to the routine. If the segment is not resident, the overlay handler reads the overlay segment from the memory image file into the specified overlay region. This destroys the previous overlay segment in that overlay region. Control then passes to the routine.

Give careful consideration to placing routines when you divide a FORTRAN IV program into a root segment and overlay regions, and subsequently divide each overlay region into overlay segments. Remember that it is illegal to call a routine located in a different overlay segment in the same overlay region, or an overlay region with a lower numeric value (as specified by the linker overlay /O:n) than the calling routine. Divide each overlay region into overlay segments that never need to be resident simultaneously.

The FORTRAN IV main program unit must be placed in the root segment.

In an overlay environment, subroutine calls and function subprogram references must refer only to one of the following:

- A FORTRAN library routine (for example, ASSIGN or DCOS).
- A FORTRAN or assembly language routine contained in the root segment.
- A FORTRAN or assembly language routine contained in the same overlay segment as the calling routine.
- A FORTRAN or assembly language routine contained in a segment whose region number is greater than that of the calling routine.

In an overlay environment, you must place the COMMON blocks so that they are resident when you reference them. Blank COMMON is always resident because it is always placed in the root segment. You must place all named COMMON either in the root segment or in the segment whose region number is lowest of all the segments that reference the COMMON block. A named COMMON block cannot be referenced by two different segments in the same region unless the COMMON block appears

in a segment of a lower region number. The linker automatically places a COMMON block into the root segment if it is referenced by the FORTRAN main program or by a subprogram that is located in the root segment. Otherwise, the linker places a COMMON block in the first segment encountered in the linker command string that references that COMMON block.

All COMMON blocks that are data-initialized (by use of DATA statements) must be so initialized in the segment in which they are placed.

The entire overlay initialization process is handled by LINK. The command format outlined below (and further explained in the linker chapter of the RT-11 System User's Guide or the RSTS/E FORTRAN IV Utilities Manual) is used to describe the overlay structure to the linker. LINK combines the run-time overlay handler with the user program, making the overlay process completely transparent to the user's program.

The size of the overlay region is automatically computed to be large enough to contain the largest overlay segment in that overlay region.

The root segment and all overlay segments are contained in the memory image file generated by LINK.

Two options are used to specify the overlay structure to LINK. The overlay option is of the form:

```
/O:n
```

where n is an octal number specifying the overlay region number. The command continuation option has two forms:

```
/C    and //
```

/C at the end of each continuation line allows the user to continue long command strings on the next line of input. // is used at the end of the first line and again at the end of the last line of input (see Table 1-8).

The first line of the LINK overlay structure command string should contain, as the input list, all object modules that are to be included in the root segment. This line should be terminated with the /C option. The /O:n option cannot appear in the first line of the command string. If all modules that are to be placed in the root segment cannot be specified on the first command line, additional modules can be specified on subsequent command lines, each ending with a /C. The entire root segment must be specified before any overlays.

All subsequent lines of the command string should be terminated with the /O:n option (switch) specifying an overlay region and/or the /C option (switch). The presence of only the /C option (switch) specifies that this is a continuation of the previous line, and therefore a continuation of the specification of that overlay segment. The object modules on each line, or set of continuation lines, constitute an overlay segment and share the specified overlay region with all other segments in the same numeric value overlay region. All but the last line of the command string should contain the /C option (switch).

For example, assume that FORLIB has been built into SYSLIB and given the following overlay structure description:

1. A main program and the object module SUB1 are to occupy the root segment.

2. The object module SUB2 is to share an overlay region with the object module SUB3 (never coresident).

3. The object modules SUB4 and SUB5 are to share a second overlay region with the object modules SUB6 and SUB7.

The following command string could be used:

RT-11

```
.LINK/PROMPT/EXE:LOAD MAIN+SUB1
*SUB2/O:1/C
*SUB3/O:1/C
*SUB4/O:2/C
*SUB5/C
*SUB6/O:2/C
*SUB7//
```

RSTS/E

```
RUN $LINK
*LOAD=MAIN,SUB1/C
*SUB2/O:1/C
*SUB3/O:1/C
*SUB4/O:2/C
*SUB5/C
*SUB6/O:2/C
*SUB7
```

1.4.2 Extended Memory Overlays (RT-11 Only)

You can use LINK to create an overlay structure that uses extended memory for privileged or virtual FORTRAN jobs. You will need an XM monitor and a hardware configuration that includes a Memory Management Unit to run a program having overlays in extended memory, but you can link such a program on any RT-11 system.

The extended-memory overlay structure is different from the low-memory overlay structure in that extended-memory overlays can reside concurrently in extended memory. This difference allows for speedier execution because, once a program is read in, it requires fewer I/O transfers with the auxiliary mass-storage volume. In fact, if all program data is resident, and the program is loaded, the program may be able to run without an auxiliary mass-storage volume.

Note that you must observe with extended-memory overlays the same restrictions that apply to low-memory overlays, especially those pertaining to return paths.

The following command string illustrates the use of extended-memory overlays to create a privileged FORTRAN job instead of the low-memory overlays used in the Section 1.4.1 example:

```
.LINK/PROMPT/EXE:LOAD   MAIN+SUB1
*SUB2/V:1/C
*SUB3/V:1/C
*SUB4/V:2/C
*SUB5/C
*SUB6/V:2/C
*SUB7//
```

Refer to the RT-11 System User's Guide for more information on Extended Memory Overlays.

1.4.3 Stand-Alone FORTRAN

You can develop FORTRAN programs under the RT-11 or RSTS/E FORTRAN IV systems and receive output in an absolute binary format for execution on a satellite machine with minimum peripherals. The satellite machine needs only a minimum of 4K words of memory and only a paper-tape reader or a serial line unit for program loading.

You can also use the stand-alone FORTRAN capability to construct Read-Only Memory (ROM) applications programs. See Section 2.5.4 for more details on ROM environments.

When operating in the stand-alone environment, the terminal is the only input/output device supported by FORTRAN-level input/output. Other devices or equipment interfaces can be supported by appropriate user-written assembly language subroutines.

To generate a stand-alone program, the source program units should be compiled as usual. At link time, special options are specified to generate a stand-alone program. The /L option must be included in the LINK command string to cause an absolute binary format (LDA) output file to be generated. The /I option must also be given to allow a special module to be requested from the FORTRAN IV Library. This module is:

```
$SIMRT      FORTRAN IV system simulator
```

Since the library used must reflect the hardware arithmetic options available on the satellite machine, you must be careful to use the proper FORTRAN IV Library. Hence, the system default library (SY:FORLIB.OBJ), which is used when the /F option (switch) is specified to LINK, may not be appropriate. Consult with the system manager (RSTS/E users) or the FORTRAN IV Installation Guide (RT-11 users) for information on the various libraries.

The following command sequence generates a file, LOAD.LDA, which can be punched on paper tape and loaded, using the Absolute Loader, on any PDP-11.

RT-11

```
.LINK/LDA/INCLUDE/EXE:LOAD MAIN,SUBS
Library search? $SIMRT
Library search? RET
```

RSTS/E

```
RUN $LINK
*LOAD,LP:=MAIN,SUBS/F/L/I
Library search? $SIMRT
*^C
READY
```

See Appendix E for further information on stand-alone FORTRAN capabilities.

1.4.4 FORTRAN Programs Run as Virtual Jobs (RT-11 Only)

You can develop FORTRAN programs which can access a full 32K words of address space, and which you can run as virtual jobs. (Virtual jobs cannot access the I/O page.) When a FORTRAN job becomes a virtual job, the FORTRAN OTS initialization code uses the special features of the .SETTOP programmed request to allocate a full 32K words of address space. The initialization code then places the OTS work area in extended memory at the high limit returned by the .SETTOP request. This allocation method differs from that of privileged FORTRAN jobs. Even if they use extended memory overlays, privileged jobs allocate all the free space in low memory for the OTS work area.

To make a FORTRAN program a virtual job, you compile the source-program units as usual, but then at link time, you specify special options.

You specify the /INCLUDE (/I) option to allow a special module to be requested from the FORTRAN IV Library. This module is: $QBLK, which is the FORTRAN IV Data Area definition for Queue Elements. Next you specify the /XM option or the /V option (specified on the first line of input to link) to allow the special .SETTOP features of the XM monitor to be enabled.

The following command sequence generates a file, VIRFOR.SAV, which can be run as a virtual job.

```
.LINK/EXE:VIRFOR/INCLUDE/XM    MAIN+SUBS
Library search?  $QBLK
Library search?  RET
```

For more information on virtual and privileged jobs, see the RT-11 System User's Guide, the RT-11 Software Support Manual, and the RT-11 Programmer's Reference Manual.

NOTE

The module $QBLK sets the virtual bit in the Job Status Word through an Asect; therefore, effect all other changes to the Job Status Word at runtime with IPEEK and IPOKE System Functions.

1.4.5 Unformatted Byte I/O

An optional module, named UIOBYT.OBJ, is included in the distribution kit. If this module is placed in the FORTRAN IV OTS Library, each byte element in an unformatted I/O statement is transferred as a byte rather than as a word. UIOBYT.OBJ can be used to save disk space when you do unformatted byte I/O. Select one of the following procedures, as appropriate for your system.

To add this module to the FORTRAN IV OTS Library (FORLIB), type:

```
.R LIBR
*FORLIB[-1]=FORLIB,UIOBYT/U/G
Global?  $ERRS
Global?  $ERRTB
Global?  RET
* ^C
.
```

If FORLIB is incorporated in SYSLIB, type:

```
.R LIBR
*SYSLIB[-1]=SYSLIB,UIOBYT/U/G
Global?  $ERRS
Global?  $ERRTB
Global?  $OVRH
Global?  RET
* ^C
.
```

1.4.6 Smaller Execution-Time Programs

The default error message module included from the FORTRAN library when linking FORTRAN programs contains the ASCII text corresponding to each possible object time error. However, the FORTRAN library also incorporates an alternate error message module which contains no text; when using this module, when a object time error occurs, only the error number is printed. The resulting core savings in using the shorter form is approximately 850 words; the corresponding on-disk savings in the size of each .SAV or .REL file is 3-4 blocks. (The core savings can be particularly significant when generating programs to be run in the foreground.) The shorter error message module can be included when linking a FORTRAN program as follows:

RT-11	Filespecs	RSTS/E
.LINK/INCLUDE	MAIN	Short error message checking is
Library search?	$SHORT	an OTS generation option.
Library search?	(RET)	Refer to the RSTS/E Installation Guide/Release Notes for information.

1.5 EXECUTION PROCEDURES

Section 1.5.1 describes program execution under the RT-11 operating system. Section 1.5.2 describes execution under the RSTS/E operating system.

1.5.1 Execution Under RT-11

Use the monitor RUN command to start execution of the memory image file generated by LINK. The command:

```
.RUN dev:filespec
```

causes the file on the device (DEV:) to be loaded into memory and executed. Filespec.sav is the file name specification as described in Section 1.1.1.

The following example takes three FORTRAN source files containing a main program and several subroutines through the procedures necessary to compile, link, and execute that program:

```
.FORTRAN/LIST  MAIN+SUB,SUBT
.LINK/MAP  MAIN,SUBT
.RUN MAIN
```

This searches SYSLIB when any undefined references are present. (The FORTRAN OTS is assumed to have been incorporated into SY6LIB upon system installation.)

1.5.2 Execution Under RSTS/E

The following command starts execution of the memory image file generated by link:

```
RUN filespec
```

Where no extension is specified in filespec, the following search occurs:

> Beginning with the first Run-Time System (RTS) in the monitor's RTS list[1] and progressing sequentially through the list, the monitor will append each RTS's default "executable file" extension to the file-name given in filespec and search for the file in the appropriate directory. When a satisfactory extension is found, the associated RTS is used to execute the file.

For example, a user has two files on his account,

> SAMPLE.BAC and
> SAMPLE.SAV

and the default RTS is BASIC-PLUS.

Typing

```
RUN SAMPLE
```

will cause SAMPLE.BAC to begin execution.

To execute SAMPLE.SAV, the extension must be given explicitly:

```
RUN SAMPLE.SAV
```

At the start of execution of a FORTRAN program, an internal initialization routine will determine exactly how much memory is required by the program from two parameters:

1. /N:xxx switch (number of logical units)

2. /R:xxx switch (maximum record length)

If necessary, additional memory is acquired to meet other requirements.

1.6 DEBUGGING A FORTRAN IV PROGRAM

The "debug line" capability of FORTRAN IV is effective in debugging because it allows a FORTRAN statement to be conditionally compiled. Here are some suggestions for using that capability.

Try to locate the statement in error by typing out intermediate values and results. Place a "D" in column one of each source line you have added for debugging purposes. These lines will not be compiled unless you specify the debug option (/D for RSTS/E, and the keyboard monitor command option /ONDEBUG for RT-11) in the compiler command string. The program can be recompiled without the /D option after the problem has been corrected. All the debugging statements will be treated as comments.

Use the operating system's ODT debugging aid for in-line code.

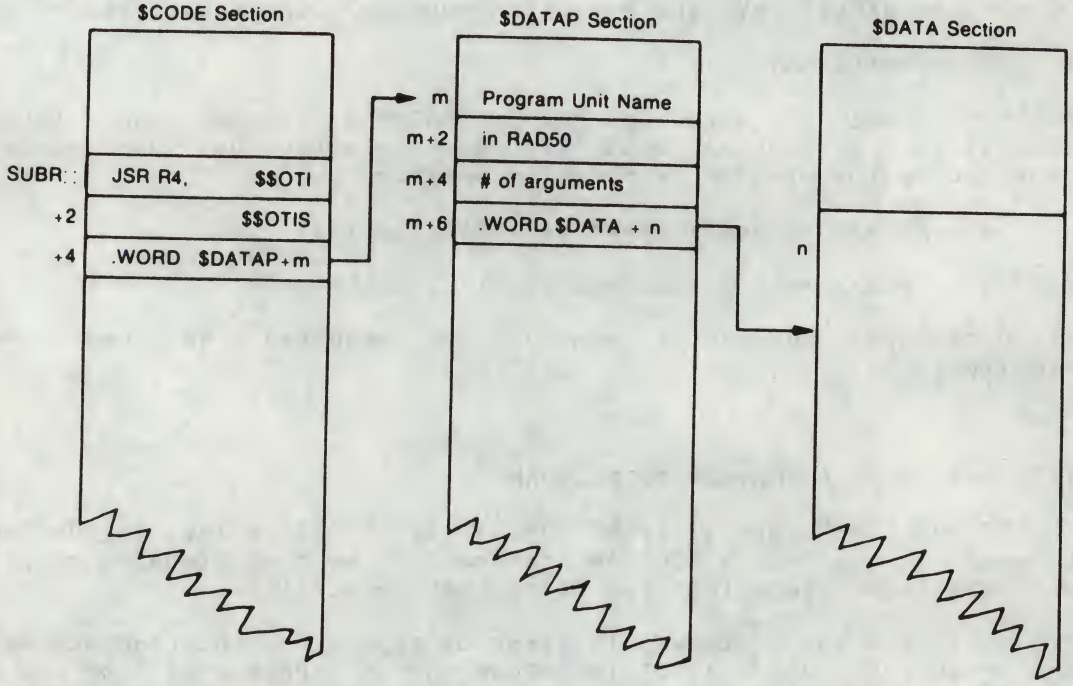
---

1. The ordering of RTS in the list will be displayed by typing the SYSTAT/R command to RSTS/E. The resulting display is the entire contents, in order, of the monitor's RTS list with associated executable file extensions.

ODT debugging requires the generated code listing for the program (see Section 1.2.2, /L or /SHOW listing control option). The resulting numbers printed in the left margin of the listing are the octal offsets of the listed machine instructions within the local PSECT $CODE.

Note that only one base address is listed for $CODE in the LINK map when multiple program units are present. To find the base of $CODE for a specific subprogram unit, locate the address of the entry point to the unit.

The variables and data items referenced symbolically in the generated code listing are located in the PSECT $DATA at the offsets indicated by the storage map section of the compiler. All local $DATA sections are formed into a single allocation on the link map when multiple programs units are linked to form a single executable program. To find the base address section for a particular program unit, examine the word at offset 4 in the unit's $CODE section. This value will be the address of the base of the unit's pure data section ($DATAP). Examining the word at offset 6 from the $DATAP section will give the base address of the associated $DATA impure section. See Figure 1-3.



Note: Only one of the following will be found in any program or subprogram unit.
MAIN:: JSR R4, $OTI - threaded main program
SUBR:: JSR R4, $OTIS - threaded SUBR or FUNCT
MAIN:: JSR R4, $$OTI - inline main program
SUBR:: JSR R4, $$OTIS - inline SUBR or FUNCT

Figure 1-3 Finding the Base Address Section

CHAPTER 2

FORTRAN IV OPERATING ENVIRONMENT

2.1 FORTRAN IV OBJECT-TIME SYSTEM

The PDP-11 FORTRAN IV Object-Time System (OTS) provides the user with a library of common sequences of PDP-11 machine instructions invoked by compiled FORTRAN programs. OTS consists of many small functional modules from which the compiler selects only those required to implement the FORTRAN program. The required sequences are integrated with the compiler-generated code during linkage to form the executable program. For example, if the user program performs only sequential access, formatted I/O, none of the direct access I/O routines is included.

The FORTRAN IV OTS comprises the following:

- Mathematics routines, including the FORTRAN IV Library functions and other arithmetic routines (for example, floating-point routines)
- Miscellaneous utility routines (RANDU, DATE, SETERR, etc.)
- Routines that handle various types of FORTRAN I/O
- Error-handling routines that process arithmetic errors, I/O errors, and system errors
- Miscellaneous routines required by the compiled code

2.2 OBJECT CODE

The FORTRAN IV compiler translates programs written in the symbolic PDP-11 FORTRAN language into "object code" (machine language). The resulting object modules, combined with required modules from the FORTRAN IV OTS, form executable programs of PDP-11 machine instructions.

The compiler produces two distinctly different types of object programs by generating either threaded code or in-line code.

When you select in-line code (through the options /CODE: [/I:] EAE, EIS, or FIS), the compiler produces the one-to-one PDP-11 machine instructions required for the specific arithmetic hardware in the system configuration. Symbolic FORTRAN library routines are referenced to perform only those functions that cannot be achieved in short sequences of machine instructions. A program compiled through an in-line code option produces an object program that conforms specifically to the type of hardware selected at compilation time.

When threaded code is generated (through the /CODE:THR [/I:THR] option), the object program produced uses a symbolic library routine to perform each operation required for program execution; the executable program consists of a "threaded" list of the addresses of library routines and appropriate operand addresses. This type of code generation produces an object module that operates independently of hardware arithmetic configuration. It may be combined with any of the FORTRAN IV OTS libraries to produce a valid executable program for each type of arithmetic hardware without any need for recompilation.

Consider the following when you decide whether to use in-line or threaded code.

When the program does not contain REAL*4, REAL*8, or COMPLEX*8 arithmetic operations,

- In-line code always executes faster than threaded.
- The differences in size between in-line and threaded programs are slight.

When the program contains large amounts of REAL*4, REAL*8, and COMPLEX*8 arithmetic (scientific computation),

- Threaded code is much smaller than in-line code.
- Execution speed is nearly the same for both.

See Table 2-1.

NOTE

Although the above relationships are generally true, they do vary from program to program. Therefore, DIGITAL recommends that production programs be compiled and tested with both in-line and threaded code. This procedure will allow you to determine the best type of code in terms of size and speed, for your application.

Table 2-1
Comparison of Threaded and In-line Code
for the Statement: I=J*K+REAL

Threaded Code	In-line Code		
	For FIS	For EIS	For EAE
MOI$MS J	MOV J,R1	MOV J,R1	MOV #$EAE,R5[1]
MUI$MS K	MUL K,R1	MUL K,R1	MOV J,(R5)+ MOV K,@R5
	MOV R1,-(SP)	MOV R1,-(SP)	MOV -(R5),-(SP)
CFI$	JSR PC,$CVTIF	JSR PC,$CVTIF	JSR PC,$CVTIF

ADF$MS REAL[2]	MOV REAL+2,-(SP)
	MOV REAL,-(SP)
	FADD SP
		MOV REAL+2,-(SP)
		MOV REAL,-(SP)
		JSR PC,$ADDF
			MOV REAL+2,-(SP)
			MOV REAL,-(SP)
			JSR PC,$ADDF

CIF$	JSR PC,$CVTFI	JSR PC,$CVTFI	JSR PC,$CVTFI
MOI$SM I	MOV (SP)+,I	MOV (SP)+,I	MOV (SP)+,I

1. $EAE represents the address of the KE11-A (or -B) accumulator register (AC).

2. Note that the threaded code sequence for this floating-point addition requires only two words of memory plus the size of the ADF$MS routine, whereas the in-line code uses five words (FIS), four words (EIS), or six word (EAE). This demonstrates the savings in storage in using threaded code for floating-point operations.

2.2.1 Processor-Defined Functions

The compiler generates code in line for the following processor-defined functions (PDFs):

Function	Definition
IABS(I)	Integer absolute value
IDIM(I,J)	Integer positive difference
ISIGN(I,J)	Integer transfer of sign
MOD(I,J)	Integer remainder
MIN0(I,J)	Integer minimum of integer list
MAX0(I,J)	Integer maximum from integer list
IFIX(A)	Real to integer conversion
FLOAT(I)	Integer to real conversion
REAL(C)	Complex to real conversion, obtain real part
DBLE(A)	Real to double conversion
SNGL(A)	Double to real conversion

Since the code for a PDF is generated by the compiler, no global reference to the function name is produced. A problem could arise when the function call is to be interpreted as a call to a user-written routine. To force the compiler to treat the apparent PDF call as a reference to a user routine, specify the routine as external to the program with an EXTERNAL statement.

For example, when compiling the statement:

```
I = IABS(J)
```

code equivalent to the following is produced:

```
     MOV    J, I
     BPL    1$
     NEG    I
1$:  ...
```

By including the statement:

```
EXTERNAL IABS
```

code equivalent to the following will be produced:

```
.GLOBL IABS
MOV    #J,-(SP)
MOV    #1,-(SP)
MOV    SP,R5
JSR    PC,IABS
CMP    (SP)+,(SP)+
MOV    R0,I
```

2.2.2 VIRTUAL Array Options (RT-11 Only)

The VIRTUAL statement declares arrays that are assigned space outside the program's directly addressable memory and are manipulated through the virtual array facility of FORTRAN IV. The VIRTUAL statement allows arrays to be stored in large data areas that are accessed at high speed. The VIRTUAL array statement is supported for RT-11, but is not currently available under RSTS/E. See the PDP-11 FORTRAN Language Reference Manual for detailed VIRTUAL array information.

VIRTUAL arrays are limited only by the number of elements, not by the total storage available. Under RT-11, all memory above the initial 28K words is available for VIRTUAL array storage. Except for the size of physical memory, there is no limit to the number or to the total size of all VIRTUAL arrays a program can access. The maximum number of elements in a VIRTUAL array is 32767. Thus the largest LOGICAL*1 VIRTUAL array is 16K words, or 32767 bytes. The largest REAL*8 VIRTUAL array is 128K words or 262136 bytes. This is a total of 32767 elements, each of which occupies 8 bytes. The limit is 32767 elements because FORTRAN IV requires array subscripts to be positive integers.

VIRTUAL support for RT-11 uses the Program Logical Address Space (PLAS) extensions when it operates under the XM monitor. The FORTRAN OTS directly manipulates the KT-11 mapping registers to provide VIRTUAL support when operating under RT-11 single job and FB monitors.

All VIRTUAL arrays declared in a program unit are allocated in a .VSECT, a type of program section. The compiler concurrently generates a .VSECT additive type of relocation as it references the VIRTUAL array in the object program.

The .VSECT program section is unnamed and has the "concatenate" attribute. This allows the accumulating of all VIRTUAL storage requirements for a linked job or task as the concatenation of the .VSECTS.

The syntax of the VIRTUAL statement is identical to that of the DIMENSION statement and involves only substituting the keyword VIRTUAL for the keyword DIMENSION. However, there is a significant semantic difference between the two because of the limitations imposed on the DIMENSION statement. Local arrays declared by the DIMENSION statement are limited by the maximum memory available to the program. Section 2.2.6 demonstrates how to convert an existing program to use the VIRTUAL feature. The three VIRTUAL array options available in building the RT-11 OTS are:

NOVIR.OBJ
VIRP.OBJ
VIRNP.OBJ

2.2.3 NOVIR.OBJ

This module specifies no VIRTUAL array support and is intended for the user who prefers to optimize the size of FORLIB rather than use VIRTUAL array support. When NOVIR.OBJ is included in the library, all references to the VIRTUAL array routines (made by the compiler when VIRTUAL arrays are available), produce ERROR 64, Virtual Array Initialization Failure.

2.2.4 VIRP.OBJ

This module is for PLAS support and requires both the XM Monitor and EIS (or FIS or FPU) hardware for program execution. Failure to adhere to these requirements will result in run-time error #64 -- VIRTUAL array initialization failure. VIRP uses the 4K words of VIRTUAL memory addresses starting at 160000 octal -- normally the PDP-11 I/O page -- for a window. For this reason, programs using VIRP support may not reference the I/O page.

The program initialization code uses the PLAS .ALLOC directive to allocate a region of the required size from the extended memory pool. This region is a contiguous section of physical memory large enough to include all VIRTUAL arrays declared in the executable program.

NOTE

If FORTRAN is unable to allocate a region of the proper size, a FATAL FORTRAN error message results.

A window of 4K words initially maps the first 4K words of the VIRTUAL array region. When a VIRTUAL array element lies outside the window, the PLAS .REMAP directive causes a window-turn operation to allow access.

2.2.5 VIRNP.OBJ

This module provides VIRTUAL array support for the single job (SJ) and the foreground/background (FB) monitors. VIRNP supports full 11/70 22-bit addressing capability. The largest amount of VIRTUAL memory available is 4068K words (4096K minus 28K). VIRNP supports the KT11 Memory Management Unit directly, mapping the job and RT-11 to kernel

space and the VIRTUAL array to user space. The module turns on the KT11 immediately before a VIRTUAL array fetch/store, and off immediately after, thus keeping KT11 mapping overhead to a minimum.

When VIRNP support is used under the FB monitor, the foreground and the background jobs cannot use VIRTUAL arrays concurrently, because both will reference the same region of extended memory for array storage. The XM monitor and PLAS VIRTUAL support must be used when two concurrent jobs require access to VIRTUAL arrays.

2.2.6 Converting a Program to Use VIRTUAL Arrays

First, be sure to observe the usage restrictions for VIRTUAL arrays covered in the PDP-11 FORTRAN Language Reference Manual. To convert the existing program, declare the arrays by using the VIRTUAL instead of the DIMENSION statement. The program does not require additional access coding.

The following example illustrates a general, minimum-effort program conversion.

1. Identify the non-VIRTUAL arrays that are to be converted to VIRTUAL arrays.

2. Locate the DIMENSION and the type declaration statements in which these arrays are declared. Replace DIMENSION statements with equivalent VIRTUAL statements. Replace array-declarative type declaration statements with VIRTUAL statements to define the array dimensions, and remove the dimensioning information from the type declaration statements.

3. Compile the program. Observe all compilation errors; these will occur where the syntax restrictions outlined in the PDP-11 FORTRAN Language Reference Manual have been violated. In some cases, you may need to reformulate the data structures to use VIRTUAL arrays effectively.

4. Check the code to ensure that VIRTUAL array parameters are passed correctly to subprograms.

    a. If the argument list of a subprogram call includes an unsubscripted VIRTUAL array name, the argument list of the SUBROUTINE or FUNCTION statement must have an unsubscripted VIRTUAL array name in its corresponding dummy argument. This establishes access to the VIRTUAL array for the subprogram. The declaration of the VIRTUAL array in the subprogram must be dimensionally compatible with the VIRTUAL declaration in the calling program. All changes to the VIRTUAL array that occurred during subprogram execution are retained when control returns to the calling program.

When you pass entire arrays as subprogram parameters, be certain that the matching arguments are defined as both VIRTUAL or both non-VIRTUAL. Mismatches of array types are not detectable at either compilation or execution time, and the results are undefined.

b. If the argument list of a subprogram reference includes a reference to a VIRTUAL array element, the matching formal parameter in the SUBROUTINE or FUNCTION statement must be a non-VIRTUAL variable. Value assignments to the formal parameter occurring within the subprogram do not alter the stored value of the VIRTUAL array element in the calling program. To alter the value of that element, the calling program must include a separate assignment statement that references the VIRTUAL array element directly.

The following example demonstrates the process of changing non-VIRTUAL arrays to VIRTUAL arrays.

```
      DIMENSION A(1000,20)
      INTEGER*2 B(1000)
      DATA B/1000*0/
      CALL ABC(A,B,1000,20)
      WRITE(2,*)(A(I,1),I=1,1000)
      END

      SUBROUTINE ABC(X,Y,N,M)
      DIMENSION X(N,M)
      INTEGER*2 Y(N)
      DO 10, I=1,N
10    X(I,1)=Y(I)
      RETURN
      END
```

This program contains two arrays, named A and B.

Array A is declared in a DIMENSION statement and is of the default data type. Thus, substituting the keyword VIRTUAL for the keyword DIMENSION is sufficient for its conversion.

Note, however, that array B and its dimensions are declared in a type declaration statement (in the second line of the program).

To convert B into a VIRTUAL array, its declarator must be moved to a VIRTUAL statement; also, the variable B must remain in the type declaration statement, but without a dimension specification.

A and B are both passed to subroutine ABC as arrays, rather than array elements. Thus the associated subroutine parameters must also be converted to VIRTUAL arrays.

The following compiled listing illustrates the program after the first phase of the conversion.

```
FORTRAN IV      V02.5      Thu 01-May-80 00:41:38

0001          VIRTUAL A(1000,20), B(1000)
0002          INTEGER*2 B
0003          DATA B/1000*0/
0004          CALL ABC(A,B,1000,20)
0005          WRITE(2,*)(A(I,1),I=1,1000)
0006          END


FORTRAN IV      Diagnostics for Program Unit .MAIN.

In line 0003,  Error:   Usage of variable "B" invalid


FORTRAN IV      Storage Map for Program Unit .MAIN.

Local Variables, .PSECT $DATA, Size = 000006 (     3. words)

Name    Type  Offset      Name    Type  Offset      Name    Type  Offset
I       I*2   000000

VIRTUAL Arrays, Total Size = 00240200 (  41024. words)

Name      Type     Offset    --------Size--------- Dimensions
A       R*4 Vec  00000000  00234200 (  40000.) (1000,20)
B       I*2      00234200  00003720 (   1000.) (1000)

Subroutines, Functions, Statement and Processor-Defined Functions:

Name    Type    Name    Type    Name    Type    Name    Type    Name    Type
ABC      R*4


FORTRAN IV      V02.5      Thu 01-May-80 00:41:40                  PAGE 001

0001          SUBROUTINE ABC(X,Y,M,N)
0002          VIRTUAL Y(N), X(N,M)
0003          INTEGER*2 Y
0004          DO 10, I=1,N
0005    10    X(I,1)=Y(I)
0006          RETURN
0007          END


FORTRAN IV      Storage Map for Program Unit ABC

Local Variables, .PSECT $DATA, Size = 000012 (     5. words)

Name    Type  Offset      Name    Type  Offset      Name    Type  Offset
I       I*2   000010      M       I*2 @ 000004      N       I*2 @ 000006

VIRTUAL Arrays, Total Size = 00000000 (       0. words)

Name      Type     Offset    -------Size-------- Dimensions
X       R*4    @   000000    ****   (  ****  ) (N,M)
Y       I*2    @   000002    ****   (  ****  ) (N)
```

Note that the main program compilation causes an error message. DATA statements must not refer to VIRTUAL arrays. The user substitutes a DO loop to achieve the same result.

The following listing shows the program after the conversion is completed.

```
FORTRAN IV      V02.5      Thu 01-May-80 00:41:25                PAGE 001

0001          VIRTUAL A(1000,20), B(1000)
0002          INTEGER*2 B
0003          DO 5, I=1,1000
0004    5     B(I)=0
0005          CALL ABC(A,B,1000,20)
0006          WRITE(2,*)(A(I,1),I=1,1000)
0007          END


FORTRAN IV      Storage Map for Program Unit .MAIN.

Local Variables, .PSECT $DATA, Size = 000006 (     3. words)

Name    Type  Offset      Name    Type  Offset      Name    Type  Offset
I       I*2   000000

VIRTUAL Arrays, Total Size = 00240200 (  41024. words)

Name      Type    Offset    -------Size-------- Dimensions
A       R*4 Vec   00000000  00234200 (  40000.) (1000,20)
B       I*2       00234200  00003720 (   1000.) (1000)

Subroutines, Functions, Statement and Processor-Defined Functions:

Name    Type    Name    Type    Name    Type    Name    Type    Name    Type
ABC      R*4


FORTRAN IV      V02.5      Thu 01-May-80 00:41:27                PAGE 001

0001          SUBROUTINE ABC(X,Y,M,N)
0002          VIRTUAL Y(N), X(N,M)
0003          INTEGER*2 Y
0004          DO 10, I=1,N
0005    10    X(I,1)=Y(I)
0006          RETURN
0007          END


FORTRAN IV      Storage Map for Program Unit ABC

Local Variables, .PSECT $DATA, Size = 000012 (     5. words)

Name    Type  Offset      Name    Type  Offset      Name    Type  Offset
I       I*2   000010      M       I*2 @ 000004      N       I*2 @ 000006

VIRTUAL Arrays, Total Size = 00000000 (       0. words)

Name      Type     Offset    -------Size-------- Dimensions
X       R*4    @   000000    ****   (  ****  ) (N,M)
Y       I*2    @   000002    ****   (  ****  ) (N)

PDS> COPY DK0:BADVRT.LST/RT TI:
```

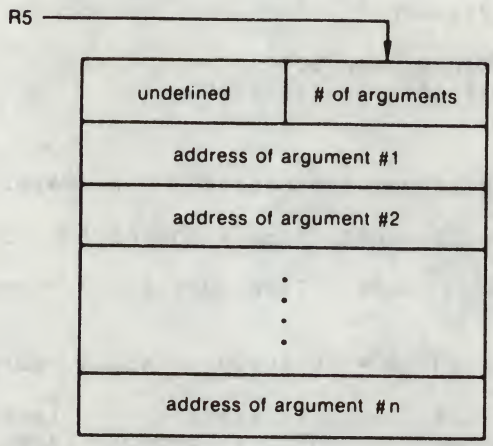
2.3 SUBPROGRAM LINKAGE

Subprogram linkage operates identically for all subprograms, including those written by the user in FORTRAN IV and in assembly language.

The following instruction is used to pass control to the subprogram:

```
JSR     PC, routine name
```

Register 5 (R5), prior to calling the subprogram, is set to the address of an argument list having the following format:



The value -1 is stored in the argument list as the address of any null arguments. Null arguments in CALL statements appear as successive commas, for example, CALL SUB (A,,B)

NOTE

Be certain that the called subprogram does not modify the argument list passed to it by the calling program.

The instruction

```
RTS     PC
```

causes control to return to the calling program.

The following is an example of argument transmission: An assembly language subroutine is written to sum all integer arguments it finds in each parameter list, and to return the result to the FORTRAN IV program as the value of a final, additional argument. The FORTRAN CALL statements that invoke this routine take the form:

```
CALL IADD(num1,num2,...,numn,isum)
```

where num1 through numn represent a variable number of integer quantities to be summed, and isum represents the variable or array element in which the sum is to be placed.

Given the following MACRO-11 subprogram:

```
        .TITLE  ADDER
        .GLOBL  IADD
IADD:   MOV     (R5)+,R0        ;GET # OF ARGUMENTS
        CLR     R1              ;PREPARE WORKING REG.
        DECB    R0              ;FIND # OF TERMS TO ADD
1$:     ADD     @(R5)+,R1       ;ADD NEXT TERM
        DECB    R0              ;DECREMENT COUNTER
        BNE     1$              ;LOOP IF NOT DONE
        MOV     R1,@(R5)+       ;RETURN RESULT
        RTS     PC              ;RETURN CONTROL
```

the sequence of FORTRAN IV calls:

```
CALL IADD(1,5,7,I)
CALL IADD(15,30,10,20,5,J)
```

would cause the variable I to be given the value 13, and the variable J to be assigned the value 80.

2.3.1 Subprogram Register Usage

A subprogram that is called by a FORTRAN IV program need not preserve any registers. However, the contents of the hardware stack must be kept so that each 'push' onto the stack is matched by a 'pop' from the stack prior to exiting from the routine.

User-written assembly language programs that call FORTRAN IV subprograms must preserve any pertinent registers before calling the FORTRAN IV routine and restore the registers, if necessary, upon return.

Function subprograms return a single result in the hardware registers. The register assignments for returning the different variable types are listed in Table 2-2.

Table 2-2
Return Value Convention for Function Subprograms

Type	Result in
INTEGER*2 LOGICAL*1	R0
INTEGER*4 LOGICAL*4	R0 -- low-order result R1 -- high-order result
REAL	R0 -- high-order result R1 -- low-order result
DOUBLE PRECISION	R0 -- highest-order result R1 -- R2 -- R3 -- lowest-order result
COMPLEX	R0 -- high-order real result R1 -- low-order real result R2 -- high-order imaginary result R3 -- low-order imaginary result

In addition, assembly language subprograms which use the FP11 Floating Point unit may be required to save and restore the FPU status. FORTRAN IV assumes that the FPU status is set by default to:

- Short floating mode (SETF)
- Short integer mode (SETI)
- Floating truncate mode

Should the assembly language routine modify these defaults, it must preserve the FPU status on entry by executing the following instruction:

```
STFPS     -(SP)
```

and restore the status (prior to returning to the calling program) by executing the instruction:

```
LDFPS     (SP)+
```

2.4 VECTORED ARRAYS

Array vectoring decreases the time necessary to reference elements of a multidimensional array by using additional memory to store the array.

Since multidimensional arrays are stored sequentially in memory, certain address calculations determine the location of individual elements. Typically, a mapping function performs this calculation. For example, to locate the element LIST(1,2,3) in an array dimensioned LIST(4,5,6), use a function equivalent to the following. This function identifies a location as an offset from the origin of the array storage.

```
(s1-1) + d1 * (s2 - 1) + d1 * d2 * (s3 - 1) =
(  0 ) + 4  * (   1  ) +  4 * 5 *  (   2  ) = 44
```

where

si = subscript i
di = dimension i

Such a mapping function requires multiplication operation(s), and some PDP-11 hardware configurations do not have the MUL instruction. The compiler can reduce execution time at the expense of memory storage by "vectoring" some arrays.

Since array vectors map only the declared dimensions of the array, you must ensure that references to arrays are within their declared bounds. A reference outside the declared bounds of a vectored array causes unpredictable results (for example a program interrupt). You must give particular attention to arrays passed to subprograms where the dimensions declared in the subprogram differ from those specified in the calling program. In such cases, two sets of vectors are created: one for the calling program and one for the subprogram. The subprogram vectors map only that portion of the array declared by the subprogram. (The PDP-11 FORTRAN Language Reference Manual contains more information on dimensions.)

A specific element in a vectored array can be located by a simplified mapping function, without the need for multiplication. Instead, a table lookup determines the location. For example, a vectored two-dimensional array B(5,6) automatically has associated with it a one-dimensional vector that would contain relative pointers to each column of array B. The location of the element B(m,n), relative to the beginning of the array, could then be computed as:

```
Vector(n) + m
```

using only addition operations. Figure 2-1 depicts the array vectoring process.

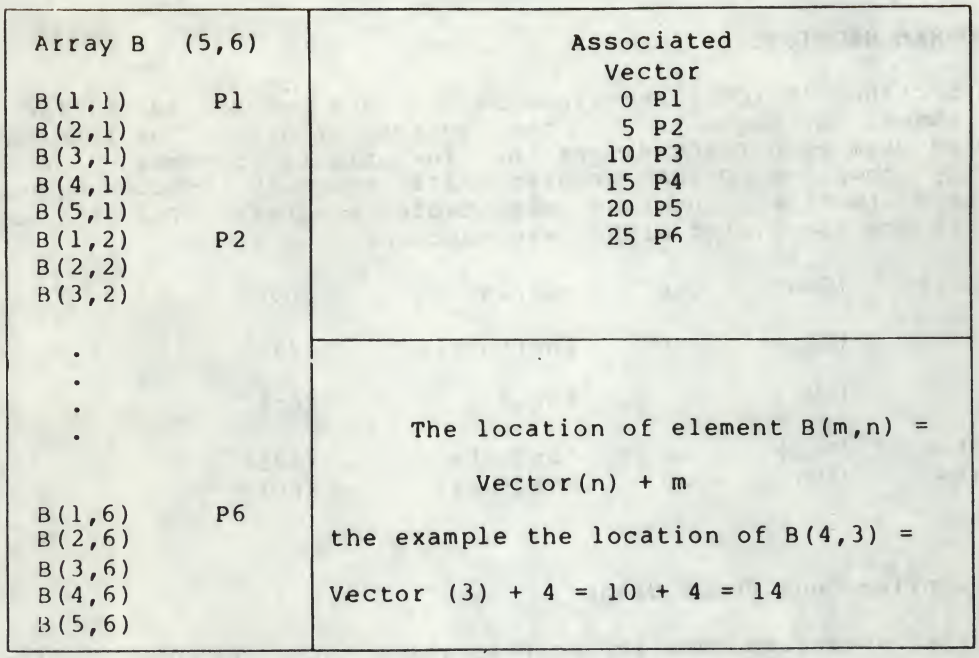


Figure 2-1 Array Vectoring

The compiler bases the decision to vector a multi-dimensional array on the computed ratio of the space required to vector the array to the total storage space it requires. The array is not vectored if this ratio is greater than 25 percent. A standard mapping function is used instead. Arrays with adjustable dimensions are never vectored. Vectored arrays are noted as such in the storage map listing.

The compiler option /NOVECTORS (/V) suppresses all array vectoring.

The amount of memory required to vector an array is the sum of all array dimensions except the first. For example, the array X(50,10,30) requires 10+30=40 words of vector table. Note that the array V(5,100) requires 100 words of vector storage, whereas the array Y(100,5) requires only 5 words of vector storage. It is good programming practice to place an array's largest dimension first when it will be vectored.

Wherever possible, vector tables are shared among several different arrays. The compiler arranges shareable vectors under the following conditions:

Arrays are in the same program unit.

For the ith dimension vector to be shared by the arrays, dimensions to the left of the ith dimension must be equivalent in each array.

For example, given the statement DIMENSION A(10,10),B(10,20), A and B share a 20-word vector for the second dimension that contains the values 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, of which the array A uses only the first 10 elements.

2.5 PROGRAM SECTIONS

Program Sections (PSECTS) contain code and data and are identified by unique names as segments of the object program. The attributes associated with each PSECT direct the Link utility to combine several separately compiled FORTRAN program units, assembly language modules, and library routines into an executable program. The following attributes are associated with these sections:

Concatenate	(CON)	or	Overlay	(OVR)
Data	(D)	or	Instruction	(I)
Global	(GBL)	or	Local	(LCL)
Relocatable	(REL)	or	Absolute	(ABS)
Read/Write	(RW)	or	Read-Only	(RO)

2.5.1 Compiled Code PSECT Usage

The compiler organizes compiled output into three program sections which have the names, attributes and contents shown in Table 2-3.

Table 2-3
Compiler Organization of Program Sections

Section Name	Attributes	Contents
$CODE	RW*, I, LCL, REL, CON	All executable code, including threaded and in-line, for a program unit
$DATAP	RW*, D, LCL, REL, CON	Pure data (for example, constants, FORMATs, array vectors), which cannot change during program execution
$DATA	RW, D, LCL, REL, CON	Impure data, variables, temporary storage, and arrays used in the FORTRAN program

* The RO/RW attribute for sections $CODE and $DATAP is controlled by the compiler /Z command option. See Section 1.2.2.

When named PSECTS with the CON attribute are concatenated by the LINK utility, all PSECTS with the same name are allocated together beginning at the same address. The length of the resulting PSECT is the sum of the individual sections so defined.

2.5.2 Common Block PSECT Usage

FORTRAN COMMON storage is placed in named PSECTs. The PSECT name is identical to the COMMON block name specified in the FORTRAN program. PSECTs used for COMMON storage are given the attributes RW, D, GBL, REL, OVR.

For example, the statement:

```
COMMON /X/ A,B,C
```

produces the equivalent of the following MACRO-11 code:

```
     .PSECT  X, RW, D, GBL, REL, OVR
A:   .BLKW   2
B:   .BLKW   2
C:   .BLKW   2
```

FORTRAN blank COMMON uses the section name .$$$$.; thus the statement:

```
COMMON C,B  /X/ A
```

produces the equivalent of:

```
     .PSECT  .$$$$., RW, D, GBL, REL, OVR
C:   .BLKW   2
B:   .BLKW   2
     .PSECT  X, RW, D, GBL, REL, OVR
A:   .BLKW   2
```

When named PSECTs with the OVR attribute are combined by the LINK utility, all PSECTs with the same name are allocated together beginning at the same address. The resulting PSECT has a length that is the maximum of the individual sections so combined.

2.5.3 OTS Library PSECT Usage

Modules included in the OTS library and referenced by the compiled program are segmented into five program sections, as shown in Table 2-4.

Table 2-4
Organization of OTS Library Modules

Section Name	Attributes	Contents
OTS$I	RW, I, LCL, REL, CON	All pure code and data for the module
OTS$P	RW, D, GBL, REL, OVR	Pure tables of addresses of other OTS library modules
OTS$D	RW, D, LCL, REL, CON	Pure data referenced by the module
OTS$S	RW, D, LCL, REL, CON	Scratch storage referenced by the module
OTS$O	RW, I, LCL, REL, CON	OTS routines sensitive to USR swapping

2.5.4 Ordering of PSECTs in Executable Programs

The order in which program sections are allocated in the executable program is controlled by the order in which they are first presented to the LINK utility.

Applications that are sensitive to this ordering typically separate those sections that contain read-only information (such as executable code and pure data) from impure sections containing variables.

The main program unit of a FORTRAN program (normally the first object program in sequence presented to LINK) declares the following PSECT ordering:

Section Name	Attributes
OTS$I	RW, I, LCL, REL, CON
OTS$P	RW, D, GBL, REL, OVR
SYS$I	RW, I, LCL, REL, CON
USER$I	RW, I, LCL, REL, CON
$CODE	RW, I, LCL, REL, CON
OTS$O	RW, I, LCL, REL, CON
SYS$O	RW, I, LCL, REL, CON
$DATAP	RW, D, LCL, REL, CON
OTS$D	RW, D, LCL, REL, CON
OTS$S	RW, D, LCL, REL, CON
SYS$S	RW, D, LCL, REL, CON
$DATA	RW, D, LCL, REL, CON
USER$D	RW, D, LCL, REL, CON
.$$$$.	RW, D, GBL, REL, OVR
Other COMMON Blocks	RW, D, GBL, REL, OVR

In the RT-11 environment, the User Service Routine (USR) may swap over pure code, but must not be loaded over constants or impure data that may be passed as arguments to it.

The above ordering, collects all pure sections before impure data in memory and USR may safely swap over sections $CODE, OTS$I, OTSP, SYSI, and USER$I.

Assembly-language routines used in applications sensitive to PSECT ordering, should use the same program sections as output by the compiler for this purpose. That is, place pure code and read-only data in section USER$I, and all impure storage in section USER$D. This will ensure that the assembly language routines will participate in the separation of code and data.

Note that the ordering of PSECTs in an overlay program will follow the guidelines herein for each overlay segment (for example, the root segment will contain pure sections followed by impure, and each overlay segment will have a similar separation of pure and impure internal to its structure).

In overlay environments, PSECTs with the GBL attribute will be allocated in the root segment if they are referenced by more than one overlay segment in the same region.

To build an application based on read-only memory (ROM), include sections $CODE, OTS$I, OTSP, SYSI, USER$I, $DATAP, OTS$O, and OTS$D in the read-only memory. Use the "round section" capability of the LINK utility to increase the size of the section OTS$D, to round the base address of section OTS$S up to the first available read-write memory location following the ROM.

ROM-based applications created in this manner must obey these programming restrictions:

- Variables or arrays may not be initialized with a DATA statement. Use assignment statements instead.
- Formatted READ statements may not transfer ASCII data into Hollerith fields in the FORMAT statement itself. Instead, place the data in a variable or array.

The PSECT ordering specified by the compiler may not be suitable for some applications. The user can define ordering by creating a MACRO-11 source file that contains only PSECT declarations. The order in which the declarations appear in this file will be the order in which the sections will be allocated in the executable program. Be sure that the attributes specified in this file agree with those assigned by the compiler.

The MACRO-11 source file containing the desired ordering must be assembled and used as the first input file in the LINK command.

The above technique allows the use of data-initialized COMMON blocks in ROM applications (if the program does not attempt to modify the values in these COMMON blocks, which will be placed in read-only memory). The new ordering defined by the MACRO-11 source file should then be:

```
.PSECT      OTS$I       RW, I, LCL, REL, CON
.PSECT      OTS$P       RW, D, GBL, REL, OVR
.PSECT      SYS$I       RW, I, LCL, REL, CON
.PSECT      USER$I      RW, I, LCL, REL, CON
.PSECT      $CODE       RW, I, LCL, REL, CON
.PSECT      OTS$O       RW, I, LCL, REL, CON
.PSECT      SYS$O       RW, I, LCL, REL, CON
.PSECT      $DATAP      RW, D, LCL, REL, CON
.PSECT      com1        RW, D, GBL, REL, OVR
.PSECT      com2        RW, D, GBL, REL, OVR
  .           .           .                .
  .           .           .                .
  .           .           .                .
  .           .           .                .
.PSECT      comn        RW, D, GBL, REL, OVR
.PSECT      OTS$D       RW, D, LCL, REL, CON
.PSECT      OTS$S       RW, D, LCL, REL, CON
.PSECT      SYS$S       RW, D, LCL, REL, CON
.PSECT      $DATA       RW, D, LCL, REL, CON
.PSECT      USER$D      RW, D, LCL, REL, CON
.PSECT      .$$$$.      RW, D, GBL, REL, OVR
```

where com1, com2, ..., comn represent the names of the read-only COMMON blocks that will contain pure data.

For further information on ROM applications, see the PROM/RT-11 User's Guide.

2.6 TRACEBACK FEATURE

The traceback feature included in RT-11, RSTS/E FORTRAN IV fatal runtime error messages locates the actual program unit and line number of a runtime error. Immediately following the error message, the error handler lists the line number and program unit name in which the

error occurred. If the program unit is a SUBROUTINE or FUNCTION subprogram, the error handler traces back to the calling program unit and displays the name of that program unit and the line number where the call occurred (see Figure 2-2). This process continues until the calling sequence has been traced back to a specific line number in the main program. This allows an exact determination of the location of an error, even if the error occurs in a deeply nested subroutine.

```
0001      A=0.0
0002      CALL SUB1(A)
0003      CALL EXIT
0004      END

0001      SUBROUTINE SUB1(B)
0002      CALL SUB2 (B)
0003      RETURN
0004      END

0001      SUBROUTINE SUB2(C)
0002      CALL SUB3 (C)
0003      RETURN
0004      END

0001      SUBROUTINE SUB3(D)
0002      E=1.0
0003      F=E/D
0004      RETURN
0005      END

          Traceback of Fatal Error:

                    ?ERR 12 FLOATING ZERO DIVIDE

                    IN   ROUTINE "SUB3   "   LINE 3
                    FROM ROUTINE "SUB2   "   LINE ?
                    FROM ROUTINE "SUB1   "   LINE 2
                    FROM ROUTINE ".MAIN."    LINE 2
```

Figure 2-2 The Traceback Feature

Note in Figure 2-2 that the line number in the traceback of routine 'SUB2' is simply a question mark (?). This is because the module was compiled with the /NOLINENUMBERS option (/S) in effect (see Section 1.2.2).

2.7 RUN-TIME MEMORY ORGANIZATION (RT-11 ONLY)

The run-time memory organization in both a USR (User Service Routines) swapping system and a USR resident system operates as shown in Figure 2-3. When user-written interrupt-handling routines are linked with a FORTRAN IV program, avoid USR swapping over the interrupt routines and any associated data. The USR is swapped in above the interrupt vectors at location $$OTSI, and extends about 2K words from that point (see Figure 2-3). Interrupt routines must therefore be loaded above the area used for USR swapping when the USR will be actively swapped. Unless explicitly disabled by the compiler option /NOSWAP (/U), the USR is actively swapped.

Some run-time memory segments are fixed in size. These include the resident monitor, the OTS work area, the stack and interrupt vector areas, and, in the USR resident system, the User Service Routine area.

Other run-time memory segments vary in length in accordance with the length of the user program. The device handlers, channel tables, and I/O buffers are allocated space dynamically. Only those handlers needed for currently active devices are resident. I/O buffers are allocated and deallocated as required.

However, there is a limit to the amount of space that can be allocated to the varying-length memory segments. In an 8K swapping system, approximately 5K words are available; in an 8K resident system, the total is approximately 3K words.

If a large FORTRAN IV program cannot be run in the amount of memory available, reduce the size of a run-time memory segment to allow successful program execution. Use overlay capabilities (see Section 1.4.1) to reduce the amount of memory needed for the user program. Minimize the number of different physical devices used for I/O to reduce the number of handlers that must be resident. When the program finishes I/O to a file, close it by using the CALL CLOSE routine (see Section B.4) or the CLOSE statement, thereby reallocating the buffer space to any new file to be opened.

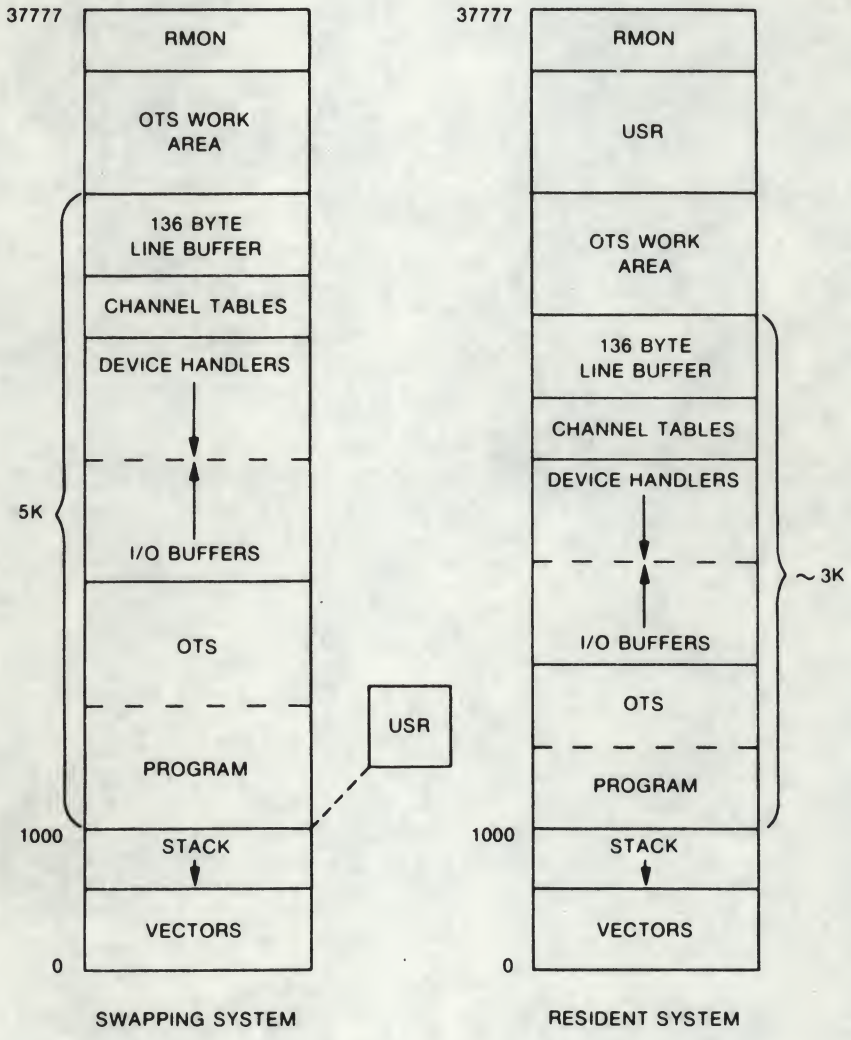


Figure 2-3 RT-11 8K System Run-time Memory Organization

CHAPTER 3

FORTRAN IV SPECIFIC CHARACTERISTICS

This chapter applies specifically to the FORTRAN IV language as it operates under the RT-11 and RSTS/E systems. The material presented here both relaxes the restrictions imposed by the PDP-11 FORTRAN Language Reference Manual and provides additional material essential to RT-11 and RSTS/E users but not covered in the Manual.

Note that these deviations from the FORTRAN syntax requirements of the manual apply only to RT-11 and RSTS/E. Your resulting program cannot be freely transported to another operating system without careful planning for that system's peculiarities and language requirements. RT-11, RSTS/E FORTRAN IV relaxes the strict statement ordering specifications of the manual and makes only the following three statement ordering requirements:

1. The first noncomment line in a subprogram must be a FUNCTION, SUBROUTINE, or BLOCK DATA statement.

2. The last line in a program unit must be an END statement.

3. Statement functions must be defined before they are referenced.

If you do not follow the statement ordering requirements of the manual, you can have a warning diagnostic included with the source listing by using the /WARNINGS option (/W).

3.1 OPEN/CLOSE STATEMENT RESTRICTIONS

Although the OPEN and CLOSE statements have an optional "ERR=label" error exit, only the following OPEN/CLOSE error conditions will cause control to be passed to "label":

Syntax error in filespec for NAME=

File not found NAME=

OPEN (UNIT=n, ...) when a file is already open on unit n

CLOSE (UNIT=n, ...) when no file is open on unit n

CLOSE (UNIT=n, DISPOSE='PRINT', ...) RSTS/E only - when an error occurs from sending the file to QUEMAN

3.1.1 Keyword Constraints

Valid keywords in the FORTRAN IV OPEN/CLOSE statements are the same for RT-11 and RSTS/E as those described in the FORTRAN Language Reference Manual.

However, since FORTRAN IV does not support some keywords and options under RT-11 and RSTS/E, certain constraints must be recognized in constructing OPEN/CLOSE statements. These constraints are shown in Table 3-1.

Table 3-1
Keyword Constraints Under RT-11 and RSTS/E

Keyword	RT-11	RSTS/E
ACCESS="APPEND"	Not supported. Results in error at compile time and at run time.	Appends data to the end of an existing file.
BUFFERCOUNT= bc	Specifies the number of buffers (one or two) to be used when outputting data on the logical unit. If not specified, one buffer is allocated.	Accepted, but ignored. The RSTS/E file system does not support multiple buffering.
BUFFERSIZE= bs	Not supported. Results in error at compile time and run time.	Same as RT-11
CARRIAGE CONTROL	Formatted files must have the attribute 'FORTRAN' or 'LIST'. Unformatted or random access files must have the attribute 'NONE'.	Same as RT-11
DISPOSE= 'PRINT'	Results in a compile-time error. Not supported.	Causes the file to be printed by the system line printer spooler under direction of Queue Manager ("QUEMAN"). The FORTRAN "QUEMAN" request includes these attributes: DEVICE= LP0 PRIORITY= 128 FILETYPE= 0 (EMB) or 1 (FTN), whichever describes the file type.
EXTENDSIZE= es	Not supported. Results in error at compile time and at run time.	Specifies the cluster size for the file. The cluster size value is the first power of two greater than or equal to es. If this value is less than pack cluster size, or if EXTEND SIZE is not specified, the pack cluster size is assumed.

(continued on next page)

Table 3-1 (Cont.)
Keyword Constraints Under RT-11 and RSTS/E

Keyword	RT-11	RSTS/E
FORM= 'FORMATTED'	Not allowed on direct access files.	Same
INITIALSIZE= is	The default initial allocation is the larger of two areas: the second largest empty area or one half of the largest area. INITIALSIZE controls the entire allocation for the file since extension is not supported.	The default initial allocation is zero blocks. The file is dynamically extended as required for write operations.
MAXREC= mr	The size of the initial allocation made for a direct access file is the larger of two parameters: INITIALSIZE or the product RECORDSIZE and MAXREC.	Same
NOSPANBLOCKS	Not supported; error	Accepted, but ignored. The RSTS/E file system does not support NOSPANBLOCKS.
RECORDSIZE	Must be specified for direct access files.	Same
SHARED	Not supported. Results in error at compile time and at run time.	Causes the file to be opened in "UPDATE" mode (mode=1) for shared access by enabling disk block locks. See also the CALL UNLOCK system routine. (App B)

3.2 SOURCE LINES

A valid RT-11, RSTS/E FORTRAN IV source line consists of the following:

1. An optional, one- to five-character, numeric statement label, followed by

2. Sufficient blanks to position the next character at column 7 (if not a continuation line) or column 6 (for continuations; a continuation signal character will be typed)

   or

   a tab character followed by any nonalphabetic character to signal continuation,

   or

   a tab character, if the line is not a continuation,

3. A valid FORTRAN statement, or the continuation of a statement, and

4. An optional comment field delimited on the left by an exclamation point (!).

Totally blank records (a source line of only a carriage return/line feed combination) are ignored on input. If a line is not totally blank, it must contain a FORTRAN statement.

3.3 VARIABLE NAMES

RT-11, RSTS/E FORTRAN IV allows variable names to exceed six characters. However, only the first six characters are significant and should be unique among all variable names in the program unit. A warning diagnostic occurs for each variable name which exceeds six characters in length. Warning diagnostics will appear in a compilation listing if the /WARNINGS option (/W) is included in the compiler command string.

3.4 INITIALIZATION OF COMMON VARIABLES

RT-11/RSTS/E FORTRAN IV allows any variables in COMMON, including blank COMMON, to be initialized in any program unit by use of the DATA statement.

3.5 CONTINUATION LINES

A line is assumed to be a continuation line if the first character following a tab on an input line to the compiler is nonalphabetic.

RT-11/RSTS/E FORTRAN IV does not place any limits on the number of continuation lines that a statement may contain.

3.6 STOP AND PAUSE STATEMENTS

The PAUSE statement causes the execution of a program to be temporarily suspended by typing the word PAUSE and the contents of the text string (if any) on the user's terminal. To resume program execution, type a carriage return.

For example, use of the FORTRAN statement:

```
PAUSE  'MOUNT A NEW TAPE'
```

causes the following line to be printed at the user's terminal:

```
PAUSE -- MOUNT A NEW TAPE
```

Execution of the STOP statement closes all files and returns control to the operating system. To terminate a program execution without printing the STOP message, use CALL EXIT. (See Section B.7.)

The STOP statement causes the following output to be printed:

```
STOP -- text
```

where text is the optional text string from the source statement.

3.7 DEVICE/FILE DEFAULT ASSIGNMENTS

The device and file name default assignments are listed in Table 3-2. The default device assignments can be changed prior to execution by using the OPEN statement or the monitor ASSIGN command.

The RT-11 monitor command:

```
.ASSIGN LP: 7
```

connects logical unit 7 to a physical device, the line printer. The device and file name assignments can be changed at execution time by use of the ASSIGN system subroutine or the FORTRAN OPEN statement. Valid logical unit numbers other than those listed below (10-99) are assigned to the system default device (public structure on RSTS/E). The default filename conventions hold for logical units not listed below, for example, unit number 49 will have a default file name of FTN49.DAT. Refer to the RT-11 System User's Guide or the RSTS/E System User's Guide.

Table 3-2
FORTRAN Logical Device Assignments

Logical Unit Number	Default Device	Default File name
1	System disk, or public structure SY:	FTN1.DAT
2	Default device	FTN2.DAT
3	Default device	FTN3.DAT
4	Default device	FTN4.DAT
5	Terminal, TT:(Input)	FTN5.DAT
6	Line printer, LP:	FTN6.DAT
7	Terminal, TT:(Output)	FTN7.DAT
8	High-speed paper tape reader, PC:	FTN8.DAT
9	High-speed paper tape punch, PC:	FTN9.DAT

Although any combination of valid logical unit numbers can be used, there is an imposed maximum number of units that can be simultaneously active. By default, six logical units can be concurrently active. The number can be changed by use of the /UNITS option (/N) in the compiler command string while compiling the main program unit (see Section 1.2.2).

A formatted READ statement of the form:

```
READ f,list
```

is equivalent to:

```
READ(1,f)list
```

For all purposes these two forms function identically. For example, assigning logical unit number 1 to the terminal, in both cases, causes input to come from the terminal.

The ACCEPT, TYPE, and PRINT statements also have similar functional analogies. Assigning devices to logical units 5, 7, and 6 affects the ACCEPT, TYPE, and PRINT statements respectively.

3.8 MAXIMUM RECORD LENGTHS

The line buffer allocated to store I/O records temporarily is by default 136 bytes. This restricts all I/O records in formatted I/O statements to a maximum of 136 characters. The size of this buffer, and consequently the maximum record length, can be changed by including the /RECORD option (/R) in the compiler command string while compiling the main program unit. The maximum size of the line buffer is 4095 bytes (7777 octal).

3.9 DIRECT ACCESS I/O

RT-11, RSTS/E FORTRAN IV allows creation and modification of direct access files.

3.9.1 DEFINE FILE Statement

The first parenthesized argument in a DEFINE FILE statement specifies the length, in records, of the direct access file being initialized. However, if the statement is part of a file creation procedure, this value may not be readily available. RT-11, RSTS/E FORTRAN IV allows some extra flexibility in this situation. Under RT-11, a file length specification of zero records causes a large contiguous file to be allocated initially and the unused portion to be automatically deallocated when the file is closed. The "END=" construction is particularly useful in this situation for determining the actual length of the file.

Under RSTS/E a file length specification of zero records causes the file to be extended dynamically as required by the highest record number referenced during program execution, if the record size is an exact multiple of 256. The DEFINE FILE statement must not be used with the OPEN statement. The OPEN statement specifies a record size in units of 2 words; whereas the DEFINE FILE statement specifies a record size in units of 1 word.

3.9.2 Creating Direct Access Files

The first I/O operation performed on a direct access file during file creation must be a WRITE operation. A READ or FIND operation under such circumstances produces a fatal error condition.

3.10 INPUT/OUTPUT FORMATS

RT-11, RSTS/E FORTRAN IV allows formatted input and output for transferring ASCII files, and unformatted and direct access input and output for transferring binary records. Note, however, that FORTRAN IV does not support ACCEPT or SEND I/O to other KB:'s.

Some run-time errors can be intercepted and control transferred to a predetermined program label by use of the ERR= parameter. This parameter can be specified in the READ, WRITE, ENCODE, or DECODE statements. Note that a count n error will become fatal on the nth occurrence of the error.

The following errors can be intercepted:

ERROR NUMBER	ERROR TYPE	MESSAGE
5	Count 3	Input conversion error
23	Fatal	Hardware I/O error
45	Fatal	Incompatible variable and format
46	Fatal	Infinite format loop

3.10.1 Formatted I/O

The formatted input/output routines read or write variable-length, formatted ASCII records. A record consists of a number of ASCII characters, transmitted under control of a format specification, followed by a record separator character(s).

On input, the parity bit of each input character is removed (set to zero); only the seven-bit ASCII character is transferred. Also, null characters (bytes of zero) are not transmitted on input.

On output to a printing device (KB:, TT:, or LP:), the record separator appended to each record consists of a carriage return character. The carriage return can be suppressed by use of the "$" format separator character in the FORMAT statement (see Chapter 6 of the PDP-11 FORTRAN Language Reference Manual). The first character of each record is deleted from the record and is interpreted as a carriage control character.

3.10.2 Unformatted I/O

When unformatted I/O routines read or write variable-length binary records, they add control and file positioning information to the user data. The control information is contained in each block of the file and allows file positioning through auxiliary I/O statements, for example, BACKSPACE. The block consists of 256 words, numbered from 0 to 255, and contains a directory describing the records that end in the block. The directory begins at word 254 and builds backward towards the start of the block. Word 255 is the number of records that end in the block; bit #15 indicates end-of-file (EOF) for the file. Each directory entry is one word long and points to the end of a record; its high order bit flags an end of file record.

As an example, assume the last three words (in octal) of block 0 of an unformatted file to be:

```
100040, 40, 2
```

and the last four words (in octal) of block 1 of the same unformatted file to be:

```
106, 56, 46, 100003
```

This is interpreted as follows:

Block 0 contains records 1, 2, and the start of record 3. Record 1 extends from block 0, byte 0 to (but not including) block 0, byte 40. Record 2 is an ENDFILE record, denoted by bit #15 being on in the end-of-record pointer; ENDFILE records have no length. Record 3 spans blocks beginning at block 0, byte 40 and extending to (but not including) block 1, byte 46. Block 1 also contains record 4 (byte 46 to 56) and record 5 (byte 56 to 106). Since the high-order bit of word 255 of block 1 is set, this file contains only 5 records.

This file format is unique to FORTRAN and may not be easily accessed by programs written in another language. To read or write a file constructed in a programming language other than FORTRAN, use formatted I/O for ASCII data and direct access I/O for binary data.

3.10.3 Direct-Access I/O

The direct access input/output routines read or write fixed-length, binary records. The logical record structure for a direct-access file is determined by the DEFINE FILE statement or the /RECORDSIZE option in the OPEN statement. The records contain only the specified data; no control information or record separators are used.

The direct access record structure is independent of the physical block size of the I/O device. However, more efficient operation results if the record size is an exact divisor or multiple of 256 words.

3.11 MIXED-MODE COMPARISONS

When comparing a single precision number to a double-precision number, the double-precision number may appear to be not equal to the single-precision number in magnitude even though they should be equal. For example:

```
DOUBLE PRECISION D

A=55.1

D=55.1D0

IF(A.LT.D)STOP
```

In the example above, A compares less than D because 55.1 is a repeating binary fraction. Before the comparison, the 24-bit fractional (mantissa) part of A is extended with 32 zero bits. These low-order 32 bits are now less than the low-order 32 bits of D, and D therefore compares greater than A. With some other values (for

example, 5.51), the single-precision value will compare greater than the double-precision value owing to the conversion rounding conventions in going from double to single precision.

3.12 FORTRAN BUFFERED I/O

FORTRAN output to sequential files other than the console terminal is sent to a 512-character buffer, which is written to the output device only when the buffer is filled or the file is closed; this process is most noticeable in a program that generates line-printer output.

If you want to force the current buffer to output to the line printer without the file being closed and reopened, include a 'REWIND' statement at each of those points.

CHAPTER 4

INCREASING FORTRAN IV PROGRAMMING EFFICIENCY

4.1 FACTORS AFFECTING PROGRAM EFFICIENCY

This chapter is directed to the programmer who is interested in minimizing program execution time or storage space requirements.

The relative efficiency of an RT-11, RSTS/E FORTRAN IV object program derives from several factors, which fall into two classes:

1. The way in which the programmer codes the source program, and

2. The way in which the compiler treats the source program.

These two factors are interrelated. Compiler optimizations can be increased by certain programming techniques in the source program. The programmer should code the source program so as to utilize those FORTRAN constructs which the compiler handles most efficiently.

Section 4.2 deals with the situations in which the compiler generates the most efficient code. Section 4.3, Programming Techniques, contains hints on improving programming efficiency.

Each topic discussed in the following section is flagged with one of the following remarks:

(space)	indicates that the primary function of the discussion is to minimize program memory requirements.
(time)	indicates that the primary concern is minimization of execution time.

A particular topic can have both designations, indicating a savings in both space and time.

4.2 INCREASING COMPILATION EFFECTIVENESS

The following 12 programming suggestions will increase compilation effectiveness.

1. Using the Optimizer effectively (space,time)

   Avoiding certain programming constructs allows the optimizer greater freedom to discover common subexpressions in source programs. Specifically, avoid the following situation:

   Usage of equivalenced and COMMON variables, and SUBROUTINE and FUNCTION dummy arguments.

2. Passing arguments to subprograms (space, time)

   To minimize overhead in FUNCTION and SUBROUTINE calls, parameters should be passed in COMMON blocks rather than standard argument lists. Variables in COMMON are handled as efficiently as local variables.

   Minimizing the number of elements in the argument list (by placing others in COMMON) reduces the time required to execute the transfer of control to the called routine.

3. Statement functions (time)

   Arithmetic and logical statement functions are implemented as internal FUNCTION subprograms. Hence, all suggestions concerning argument lists apply to statement functions also.

4. Minimizing array vector table storage (space)

   The RT-11, RSTS/E FORTRAN IV array-vectoring feature is designed to decrease the time required to compute the address associated with an element of a multidimensional array by precomputing certain of the multiplication operations involved. The values precomputed are stored in a table called the "vector" for the particular array dimension. It is desirable to minimize the space allocated to these vectors.

   The following steps can be taken by the programmer to reduce the space required for array vectors:

   - Specify the largest dimensions first in the statement that allocates the array. This minimizes the number of vector table entries, as the first dimension is never vectored. For example,

     INTEGER A(350,10) requires 10 words to vector

     INTEGER A(10,350) requires 350 words to vector

     The compiler computes a space tradeoff factor that relates the number of words required for vector storage to the number of words required to store the array. If this tradeoff is favorable (for example, the vector table is small compared to the array), the array is vectored. Therefore, the proper ordering of dimensions not only saves table space for all vectored arrays, but can also cause other arrays to be made eligible for vectoring.

   - Try to keep similar arrays dimensioned in the same order. This will cause certain arrays to share vector tables. For example:

     INTEGER A(9,4,5), B(9,4,7), C(9,8)

     all share the same two vectors, one for the second array dimension and one for the third. The vector for the second dimension will have eight elements (@ 1 word each) because C has the largest second dimension, 8. Similarly, the vector for the third dimension has seven elements.

In the general case, two arrays share a vector table for dimension i if each dimension less than i in each array is identical to the same dimension for the other array. In the example given above, arrays A, B, and C share the vector for the second dimension because each array has a first dimension equal to 9.

- Vectoring can be disabled completely by specifying the /VECTORS (/V) option in the command string to the compiler. This causes no vector tables to be generated, but the resulting program executes more slowly than with vectoring. This tradeoff can be made if array usage is not heavy in speed-critical sections of the program, or if space is the primary goal.

5. Multidimensioned array usage (time)

When using multidimensional arrays, the number of specified variable subscripts affects the time required to make the array reference. Therefore, the following steps can be taken to optimize array references:

- Use arrays with as few dimensions as possible.

- Use constant subscripts whenever possible. Constant subscripts are computed during compilation and require no extra operations at execution time.

- Make totally constant array references wherever appropriate. These references receive the highest level of optimization. For example,

```
I = 1
A(I) = 0.0
```

is not as efficient as

```
I = 1
A(1) = 0.0
```

The former case requires a run-time subscript operation; in the latter, the compiler can calculate the address of the first element of array A at compilation time.

6. Formatted input/output (space,time)

RT-11, RSTS/E FORTRAN IV precompiles and compacts FORMAT statements that are presented in the source program. This affects the space required to store the format at run time, and the speed of the input/output operations that make use of the format.

For this reason, object-time formats (for example, those formats specified in arrays rather than as FORMAT statements) are considerably less efficient.

7. Data type selection (space,time)

Because of the addressing modes of the PDP-11 processors and various optimization considerations internal to FORTRAN IV,

more efficient code can be generated for certain data types than for others. Specifically:

- Use the INTEGER data type wherever possible. RT-11, RSTS/E FORTRAN IV performs extensive optimizations on this data type.
- Use REAL*4 rather than DOUBLE PRECISION (REAL*8) wherever possible. Single-precision operations are significantly faster than double-precision, and storage space is saved.
- Avoid unnecessary mode mixing. For example:

  A = 0.0

  is preferable to

  A = 0

- Use two REAL*4 variables rather than a COMPLEX*8 if usage of COMPLEX variables in the program is not heavy. REAL*4 operations receive more optimization than COMPLEX operations.

8. Testing "flag" variables (space)

   Wherever possible, comparisons with zero should be used. Comparing any data type to a zero value is a special case that requires less executable code. An example of such a case is the following:

```
IF (I .LT.  1) GOTO 100
```

   requires more code than

```
IF (I .LE.  0) GOTO 100
```

9. *2, **2 operations (time)

   Explicitly specifying *2 when doubling the value of an expression, or **2 when squaring the value of an expression can lead to more efficient code. For example:

```
A = (B + ARRAY(C))**2
```

   is preferable to

```
A = (B + ARRAY(C)) * (B + ARRAY(C))
```

   despite the fact that (B + ARRAY(C)) is computed only once in either case. Note that this applies only to expression values; I**2 is as efficient as I*I.

10. Compilation options (space)

    To minimize the space required for program execution, the following options should be supplied to the compiler:

/NOLINENUMBERS	(/S)	to suppress line number traceback
/VECTORS	(/V)	to suppress all array vectoring

    In addition, the /I4 (/T) (two-word integer default) option should not be specified unless required.

The /NOSWAP (/U) OPTION should not be specified if it is not required (i.e., no user-written interrupt or completion routines exist in the linked program).

Specify minimal values for the /UNITS (/N) and /RECORDS (/R) compiler options to conserve space at execution time. The /UNITS (/N) value should be the number of logical units that can be concurrently active. Set the /RECORDS (/R) option to the maximum formatted record length plus two (for the carriage return/line feed combination that can accompany a record).

12. Compilation options (time)

Specify the following compiler options to optimize an object program for execution time.

/NOLINENUMBERS	(/S)	to suppress line number traceback
/NOSWAP	(/U)	to prevent the USR (RT-11 file service routines) from swapping at run time (RT-11 only; ignored under RSTS/E)

Do not specify the following option, because global array vectoring speeds program execution.

/NOVECTORS	(/V)	will disable array vectoring

4.3 PROGRAMMING TECHNIQUES

The following examples compare different programming methods. These comparisons show more efficient programming techniques available to the user. While both methods are correct for the particular operation, the technique on the right has been found more efficient than the technique on the left.

1. Make use of the increment parameter in DO loops:

```
    INEFFICIENT                    EFFICIENT

    DIMENSION A(20)                DIMENSION A(20)
    DO 100 I=1,10                  DO 100 I=2,20,2
    A(2*I)=B                       A(I)=B
100 CONTINUE                   100 CONTINUE
```

In the inefficient example, an additional calculation (2*I) is performed each time through the loop. These calculations are avoided in the efficient example by having the count incremented by two.

2. Avoid placing calculations within loops whenever possible:

```
    INEFFICIENT                    EFFICIENT

   DO 10 I=1,20                    TEMP1=B*C
   DO 20 J=1,50                    DO 10 I=1,20
20 A(J)=A(J)+I*B*C                 TEMP2=I*TEMP1
10 CONTINUE                        DO 20 J=1,50
                               20 A(J)=A(J)+TEMP2
                               10 CONTINUE
```

The calculation (B*C) within the loop of the inefficient example is evaluated 1000 times. Calculations are handled more economically when done outside the loop. In the efficient example, 980 "FLOATS" and 1979 floating multiplies were saved by performing the (B*C and I) calculations outside the loop.

3. Proper nesting of DO loops can increase speed by minimizing the loop initialization.

INEFFICIENT

```
   DIMENSION A(100,10)
   DO 60 I=1,100
   DO 60 J=1,10
60 A(I,J)=B
```

EFFICIENT

```
   DIMENSION A(100,10)
   DO 60 J=1,10
   DO 60 I=1,100
60 A(I,J)=B
```

In the first example, the inner DO loop is initialized 100 times, while in the efficient example it is only initialized 10 times.

4. The most efficient way to zero a large array, or to set each element to some value, is to equivalence it to a single-dimension array. This technique is even useful for copying large multidimensional arrays.

INEFFICIENT

```
   INTEGER A(20,100)
   DO 20 I=1,100
   DO 20 J=1,20
20 A(J,I)=0
```

EFFICIENT

```
   INTEGER A(20,100)
   REAL*8 ATEMP(500)
   EQUIVALENCE (A,ATEMP)
    DO 20 I=1,500
20 ATEMP(I)=0.0D0
```

The efficient example makes use of the eight bytes in REAL*8 and, by equivalencing, places four integers in the array and zeroes them in one operation, thus quartering the number of iterations.

5. Do as much calculation in INTEGER mode as possible.

INEFFICIENT

```
A=B+I+J
```

EFFICIENT

```
A=B+(I+J)
```

Also, do as much calculation in REAL mode when the dominant mode of an expression is DOUBLE PRECISION or COMPLEX. Calculation is most efficient in integer mode, less efficient in REAL mode, and least efficient in DOUBLE PRECISION or COMPLEX. Remember, in the absence of parentheses, evaluation generally proceeds from left to right.

6. Use COMMON to pass arguments and return results of subprograms whenever possible.

INEFFICIENT

```
CALL SUBR(A,B,C,D,E)
X=FUNCT(Y,Z)
CALL SUBR(A,B,C,D,E)
        .
        .
        .
END
SUBROUTINE SUBR(A,B,C,D,E)
        .
        .
        .
END

FUNCTION FUNCT(Y,Z)
        .
        .
        .
END
```

EFFICIENT

```
COMMON/SUBRA/A,B,C,D,E
COMMON/FUNCTA/Y,Z
        .
        .
        .
CALL SUBR
X=FUNCT()
CALL SUBR
        .
        .
        .
END
SUBROUTINE SUBR
COMMON/SUBRA/A,B,C,D,E
        .
        .
        .
END
FUNCTION FUNCT
COMMON/FUNCTA/Y,Z
        .
        .
        .
END
```

COMMON is handled more efficiently than formal argument lists. Generally, it is possible to use COMMON for argument passage if a subprogram is referenced from only one place, or if it is always referenced with the same actual arguments.

NOTE

In PDP-11 FORTRAN IV, function subprograms are not required to have arguments, but they must have empty parentheses for the compiler to recognize them as functions; for example, IGETC().

7. Avoid division within programs wherever possible.

INEFFICIENT	**EFFICIENT**
A=B/2.	A= B*.5

Multiplication is faster than division and thus saves execution time.

CHAPTER 5

CONCISE COMMAND LANGUAGE OPTION

5.1 INTRODUCTION TO THE RSTS/E FORTRAN IV CCL OPTION

The Concise Command Language (CCL) commands provide an alternative method for invoking RSTS/E system programs. CCL commands allow a user to run a system program by specifying a single command for the program to execute. The user types the CCL command and the program command on one line and enters it to the system. The system loads the program into the user's job area and writes the program command to the core common area. This operation destroys the current contents of the user's job area. The program runs, reads the command from the core common area, and executes. If an error is encountered, the program prints a related message and terminates. CCL options are available for the following system programs: FORTRAN, LINK, AND MACRO.

RSTS/E users should contact the system manager for the availability of these commands on their system.

5.2 COMMAND INTERFACE

The CCL command to invoke the FORTRAN IV Compiler has the form:

```
FOR[TRAN] command line
```

where

FOR[TRAN]	indicates that the FORTRAN command can be abbreviated to these characters (FOR).
command line	has the form: output = input/sw The output and input file name specifications are described in Section 1.1.1; the compiler switches are described in Section 1.2.1.

The command to invoke the linker, LINK, has the form:

```
LINK command line
```

where

command line	has the form: output = input/sw The output and input file name specifications and switch options are described in Section 1.3.

The command to invoke MACRO has the form:

```
MACRO command line
```

where

command line — has the form: output = input/sw
The output and input file name specifications and switch options are described in the RSTS/E FORTRAN IV Utilities Manual.

5.2.1 CCL Command Restrictions

Several switch options included in the LINK utility are not acceptable to the LINK CCL command line. These switch option restrictions do not apply to the "RUN $LINK" invocation of the linker utility but only to one line of input to the LINK CCL command. The restricted switches are the following:

/C	continue input specification on multiple lines
/E	extend PSECT
/I	include requested library modules
/M	specify stack address as global symbol (/M:n form is acceptable)
/O	indicate overlay structure
/T	specify transfer address as global symbol (/T:n form is acceptable)
/U	round PSECT
//	indefinite continuation

5.2.2 CCL Command Comparison

The following example illustrates the two methods available to the user for creating a source and assembly program, as well as linking and execution.

RSTS/E Command String	CCL
RUN $FORTRAN	FOR MAIN=MAIN,SUBR/S
*MAIN=MAIN,SUBR/S	
*^Z	
READY	READY
RUN $MACRO	MACRO MACSUB=MACSUB
*MACSUB=MACSUB	ERRORS DETECTED:0
ERRORS DETECTED:0	FREE CORE: 1024 WORDS
FREE CORE: 1024 WORDS	
*^Z	

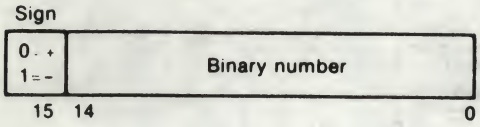
```
READY

RUN $LINK
*PROG=MAIN,MACSUB/F


*^Z

READY

RUN PROG
```

```
READY

LINK PROG=MAIN,MACSUB/F


READY

RUN PROG
```

APPENDIX A

FORTRAN DATA REPRESENTATION

A.1 INTEGER FORMAT



Integers are stored in a two's complement representation. If the /I4 (/T) compiler option (see Section 1.2.1) is used, an integer is assigned two words, although only the low-order word (for example, the word having the lower address) is significant. By default, integers will be assigned to a single storage word. Explicit length integer specifications (INTEGER*2 and INTEGER*4) will always take precedence over the setting of the /I4 (/T) option. Integer constants must lie in the range -32768 to +32767. For example:

```
+22 = 000026(octal)
 -7 = 177771(octal)
```

A.2 FLOATING-POINT FORMATS

The exponent for both two-word and four-word floating-point formats is stored in excess 128 (200(octal)) notation. Binary exponents from -128 to +127 are represented by the binary equivalents of 0 through 255 (0 through 377(octal)). Fractions are represented in sign-magnitude notation with the binary radix point to the left. Numbers are assumed to be normalized and, therefore, the most significant bit is not stored because of redundancy (this is called hidden bit normalization). This bit is assumed to be a 1 unless the exponent is 0 (corresponding to 2-128), in which case it is assumed to be 0. The value 0 is represented by two or four words of zeros. For example, +1.0 would be represented by:

```
40200
0
```

in the two-word format, or:

```
40200
0
0
0
```

in the four-word format, -5 would be:

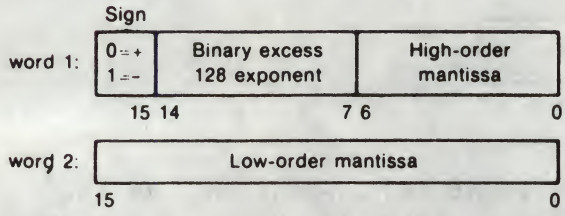
```
140640
0
```

in the two-word format, or:

```
140640
0
0
0
```

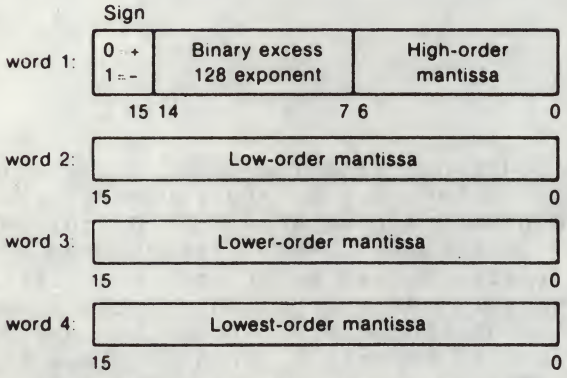
in the four-word format.

A.2.1 REAL Format (Two-Word Floating Point)



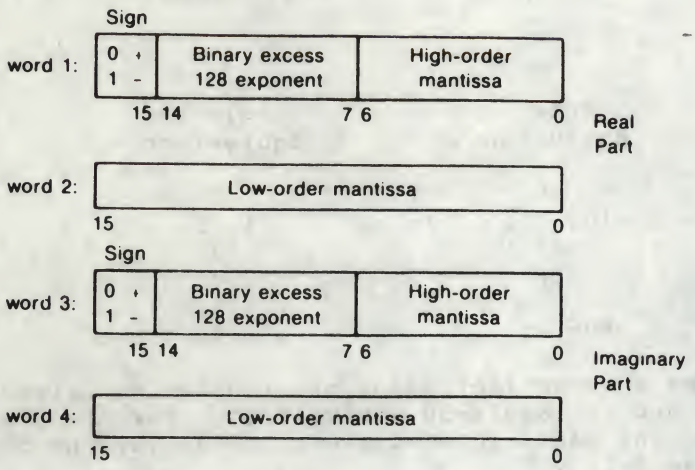
Since the high-order bit of the mantissa is always 1, it is discarded, giving an effective precision of 24 bits (or approximately 7 digits of accuracy). The magnitude range lies between approximately .29 X 10-38 and .17 X 1039.

A.2.2 DOUBLE PRECISION Format (Four-Word Floating Point)

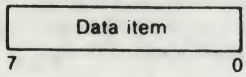


The effective precision is 56 bits (or approximately 17 decimal digits of accuracy). The magnitude range lies between .29 X 10-38 and .17 X 1039.

A.2.3 COMPLEX Format

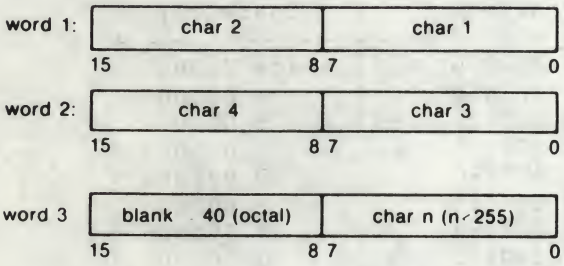


A.3 LOGICAL*1 FORMAT



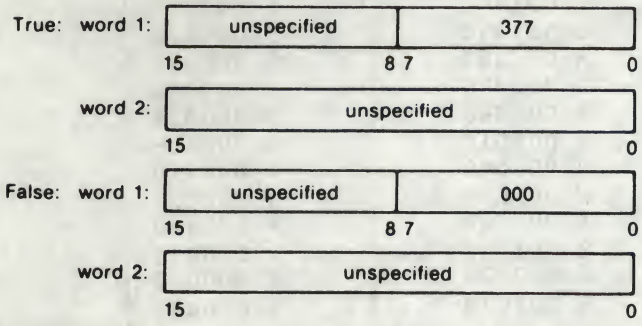
Any non-zero value is considered to have a logical value of .TRUE. The range of numbers from +127 to -128 can be represented in LOGICAL*1 format. LOGICAL*1 array elements are stored in adjacent bytes.

A.4 HOLLERITH FORMAT



Hollerith constants are stored internally, one character per byte. Hollerith values are padded on the right with blanks to fill the associated data item, if necessary.

A.5 LOGICAL FORMAT



Logical (LOGICAL*4) data items are treated as LOGICAL*1 values for use with arithmetic and logical operators. Any non-zero value in the low order byte is considered to have a logical value of true in logical expressions.

A.6 RADIX-50 FORMAT

Radix-50 character set

Character ASCII	Octal Equivalent	Radix-50 Equivalent
space	40	0
A-Z	101-132	1-32
$	44	33
.	56	34
unused		35
0-9	60-71	36-47

The following table provides a convenient means of translating between the ASCII character set and its Radix-50 equivalents. For example, given the ASCII string X2B, the Radix-50 equivalent with (arithmetic is performed in octal) is as follows:

```
X  =113000
 2 =002400
  B=000002
X2B=115402
```

Table A-1
ASCII/Radix-50 Equivalents

Single Character or First Character	Second Character	Third Character
space 000000	space 000000	space 000000
A 003100	A 000050	A 000001
B 006200	B 000120	B 000002
C 011300	C 000170	C 000003
D 014400	D 000240	D 000004
E 017500	E 000310	E 000005
F 022600	F 000360	F 000006
G 025700	G 000430	G 000007
H 031000	H 000500	H 000010
I 034100	I 000550	I 000011
J 037200	J 000620	J 000012
K 042300	K 000670	K 000013
L 045400	L 000740	L 000014
M 050500	M 001010	M 000015
N 053600	N 001060	N 000016
O 056700	O 001130	O 000017
P 062000	P 001200	P 000020
Q 065100	Q 001250	Q 000021
R 070200	R 001320	R 000022
S 073300	S 001370	S 000023
T 076400	T 001440	T 000024
U 101500	U 001510	U 000025
V 104600	V 001560	V 000026
W 107700	W 001630	W 000027
X 113000	X 001700	X 000030
Y 116100	Y 001750	Y 000031
Z 121200	Z 002020	Z 000032
$ 124300	$ 002070	$ 000033

(continued on next page)

Table A-1 (Cont.)
ASCII/Radix-50 Equivalents

Single Character or First Character	Second Character	Third Character
. 127400	. 002140	. 000034
unused 132500	unused 002210	unused 000035
0 135600	0 002260	0 000036
1 140700	1 002330	1 000037
2 144000	2 002400	2 000040
3 147100	3 002450	3 000041
4 152200	4 002520	4 000042
5 155300	5 002570	5 000043
6 160400	6 002640	6 000044
7 163500	7 002710	7 000045
8 166600	8 002760	8 000046
9 171700	9 003030	9 000047

APPENDIX B

LIBRARY SUBROUTINES

B.1 LIBRARY SUBROUTINE SUMMARY

In addition to the functions intrinsic to the FORTRAN IV system, the FORTRAN library contains the following subroutines that the user can call in the same manner as a user-written subroutine.

ASSIGN	allows specification at run time of the file name or device and file name to be associated with a FORTRAN IV logical unit number.
OPEN	(RSTS/E only) causes the specified file to be opened and associated with a particular FORTRAN IV logical unit.
CLOSE	closes the specified logical unit after writing any active buffers to the file.
DATE	returns a nine-byte string containing the ASCII representation of the current date.
IDATE	returns three integer values representing the current month, day, and year.
EXIT	terminates the execution of a program and returns control to the executive.
USEREX	allows specification of a routine to be invoked as part of program termination.
RANDU, RAN	returns a pseudo random-real number with a uniform distribution between 0 and 1.
SETERR	allows the user to set a count specifying the number of times to ignore a certain error condition.
ERRTST	allows the user program to monitor the types of errors detected during program execution.
ERRSNS	allows the user program to obtain information about the most recent error that has occurred during program execution.
UNLOCK	(RSTS/E only) unlocks the disk block currently held for update on the file open for shared access on the logical unit specified.

B.2 ASSIGN

The CALL ASSIGN statement should not be used in conjunction with the CALL OPEN statement. In proper context, the ASSIGN subroutine allows the association of device and file name information with a logical unit number. The ASSIGN call, if present, must be executed before the logical unit is opened for I/O operations (by READ or WRITE) for sequential access files, or before the associated DEFINE FILE statement for random access files. The assignment remains in effect until the end of the program or until the file is closed by CALL CLOSE or the CLOSE statement, and a new CALL ASSIGN performed. The call to ASSIGN has the general form:

```
CALL ASSIGN(n, name, icnt, mode, control, numbuf)
```

CALL ASSIGN requires only the first argument all others are optional, and if omitted, are replaced by the default values as noted in the argument descriptions. However, if any argument is to be included, all arguments that precede it must also be included.

NOTE

Under RSTS/E, any project-programmer number or protection code information supplied to CALL ASSIGN is ignored. If the ability to supply such information is desired, CALL OPEN should be used.

A description of the arguments to the ASSIGN routine follows:

n — logical unit number expressed as an integer constant or variable.

name — Hollerith or literal string containing any standard RT-11 or RSTS/E file specification. If the device is not specified, then the device remains unchanged from the default assignments. If a file name is not specified, the default names, as described in Section 3.7, are used. The three options that can be included in the file specification are:

- /N — specifies no carriage control translation. This option overrides the value of the 'control' argument.
- /C — specifies carriage control translation. This option overrides the value of the 'control' argument.
- /B:n — specifies the number of buffers, n, to use for I/O operations. The single argument, n, should be of value 1 or 2. This option overrides the value of the 'numbuf' argument. Multibuffering is not supported under RSTS/E.

If name is simply a device specification, the device is opened in a non-file-structured manner, and the device is treated in a non-file-structured manner. Indiscriminate use of this feature on directory devices such as disk or DECtape can be dangerous (for example, damage the directory structure).

icnt — specifies the number of characters in the string 'name'. If 'icnt' is zero, the string 'name' is processed until the first blank or null character is encountered. If 'icnt' is negative, program execution is temporarily suspended. A prompt character (*) is sent to the terminal, and a file name specification, with the same form as 'name' above, terminated by a carriage return, is accepted from the keyboard.

mode — specifies the method of opening the file on this unit. This argument can be one of the following:

'RDO' — the file is read only. A fatal error occurs if a FORTRAN write is attempted on this unit. If the specified file does not exist, run-time error 28 (OPEN FAILED FOR FILE) is reported.

'NEW' — a new file of the specified name is created; this file does not become permanent until the associated logical unit is closed via the CALL CLOSE routine, the CLOSE statement or program termination. If execution is aborted by typing CTRL^C, the file is not preserved.

'OLD' — the file already exists. If the specified file does not exist, run-time error 28 (OPEN FAILED FOR FILE) is reported.

'SCR' — the file is only to be used temporarily and is deleted when it is closed.

If this argument is omitted, the default is determined by the first I/O operation performed on that unit. If a WRITE operation is the first I/O operation performed on that unit, 'NEW' is assumed. If a READ operation is first, 'OLD' is assumed.

control — specifies whether carriage control translation is to occur. This argument can be one of the following:

'NC' — all characters are output exactly as specified. The record is preceded by a line feed character and followed by a carriage return character.

'CC' — the character in column one of all output records is treated as a carriage control character. (See the PDP-11 FORTRAN Language Reference Manual.)

If not specifically changed by the CALL ASSIGN subroutines, the terminal and line printer assume by default 'CC', and all other devices assume 'NC'.

numbuf — specifies the number of internal buffers to be used for the I/O operation. A value of 1 is appropriate under normal circumstances. If this argument is omitted, one internal buffer is used. Multibuffering is not supported under RSTS/E.

B.3 OPEN (RSTS/E ONLY)

The CALL OPEN statement should not be used in conjunction with the OPEN statement. The subroutine OPEN is an extension of the ASSIGN routine for RSTS/E. OPEN allows a specified file to be opened and associated with a particular FORTRAN logical unit. In the OPEN call, all arguments (except "n" and "name") are optional and will default if not specified. However, if any argument is to be included, all arguments that precede it must also be included.

A description of the arguments to the OPEN subroutine follows:

CALL OPEN (n, name, icnt, disp, control, numbuf, iotype, p, pn, prot, mode, cluster)

where

n — is the integer specification of the logical unit to be associated with the file.

name — is a variable array, or quoted string, whose contents specify the name (and possibly the project-programmer number and protection code) of the file to be opened.

icnt — is an integer value that controls the interpretation of the 'name' argument. If 'icnt' is positive, it specifies the number of characters to be taken from the 'name' argument as the file name string. If 'icnt' is zero, 'name' is scanned until the first blank or null character is encountered. If a negative value is given for 'icnt', program execution is temporarily suspended, a prompt character (*) is sent to the terminal, and a filename specification, terminated by a carriage return, is accepted from the keyboard.

disp — is a string specification of the disposition of the file on this unit. This argument can be one of the following:

'RDO' the file is read only. A fatal error occurs if a FORTRAN write is attempted on this unit. If the specified file does not exist, run-time error 28 (OPEN FAILED FOR FILE) is reported.

'NEW' a new file of the specified name is created; this file does not become permanent until the associated logical unit is closed via the CALL CLOSE routine, the CLOSE statement or program termination. If execution is aborted by typing CTRL^C, the file is not preserved.

'OLD' the file is assumed to exist. If the file is not found in the specified directory, or is protected against access by the user, a fatal error results.

'SCR' the file is only to be used temporarily and will be deleted when it is closed.

If this argument is not given, the default is set to 'NEW'.

control — is a string argument-specifies whether carriage control translation is to occur. This argument can be one of the following:

'NC' all characters are output exactly as specified. The record is preceded by a line feed character and followed by a carriage return character.

'CC' the character in column one of all output records is treated as a carriage control character. (See the PDP-11 FORTRAN Language Reference Manual.)

If not specifically changed by the OPEN routine, the user's terminal and the line printer assume by default 'CC', and all other devices assume 'NC'.

numbuf — retained for argument list compatibility with the RT-11 ASSIGN system subroutine; has no function under RSTS/E and should be omitted or given the value 1.

iotype — is a string argument that specifies the range of input/output operations to be performed on a unit. This argument can be one of the following:

'FOR' the unit is to be opened for formatted input/output.
'UNF' the unit is to be opened for unformatted input/output.
'RAN' the unit is to be opened for random access input/output.

If this argument is not specified, it defaults to 'FOR'.

p — is an integer value giving the default project code to be used (in conjunction with the "pn" argument) if no project-programmer number specification is found in "name".

pn — is an integer value giving the default programmer code to be assumed if no project-programmer specification appears in "name".

prot — is an integer value specifying the protection code to be assigned by default if no protection code indication occurs in "name". This argument takes effect only on output files.

mode — is an integer specification of the RSTS/E mode to be used on opening the file (refer to the RSTS/E Programming Manual for device-specific mode information).

cluster — is an integer specification of the cluster size to be assigned to the file to be opened. This argument only takes effect on output files (for example, files with the 'NEW' or 'SCR' attribute).

B.4 CLOSE

The user has the option of the CALL CLOSE routine and the CLOSE statement. The CALL CLOSE routine is a subset of the CLOSE statement (see the PDP-11 FORTRAN Language Reference Manual). CLOSE explicitly closes any file open on the specified logical unit. If the file was open for output, any partially filled buffers are written to the file before closing it. After the execution of CALL CLOSE, any buffers associated with the logical unit are freed for reuse and all information supplied in any previous CALL ASSIGN for the logical unit is deleted. The logical unit is thus free to be associated with another file.

An implicit CLOSE operation is performed on all open logical units when a program terminates (due to a fatal error condition, or the execution of STOP or CALL EXIT).

The format of the call is:

```
CALL CLOSE (ilun)
```

where

ilun is an integer constant, variable, array element, or expression specifying the logical unit to be closed.

B.5 DATE

The DATE subroutine can be used in a FORTRAN program to obtain the current date as set within the system. The DATE subroutine is called as follows:

```
CALL DATE(array)
```

where array is a predefined array able to contain a nine-byte string. The array specification in the call can be expressed as the array name alone:

```
CALL DATE(a)
```

in which the first three elements of the real array a are used to hold the date string, or:

```
CALL DATE(a(i))
```

which causes the nine-byte string to begin at the i(th) element of the array a.

The date is returned as a 9-byte (nine-character) string in the form:

```
dd-mmm-yy
```

where

dd is the two-digit date (with leading zero if necessary)
mmm is the three-letter month specification (all capital letters)
yy is the last two digits of the year

For example:

```
25-DEC-76
```

The 9-byte array is set to blanks, if the system date has not been set.

B.6 IDATE

IDATE returns three integer values representing the current month, day, and year. The call has the form:

```
CALL IDATE(i,j,k)
```

If the current date were March 19, 1976, the values of the integer variables upon return would be:

```
i = 3
j = 19
k = 76
```

i is returned as zero, if the system date has not been set.

B.7 EXIT

A call to the EXIT subroutine, in the form:

```
CALL EXIT
```

is equivalent to the STOP statement except that no STOP message appears on the user's terminal. Use of the EXIT statement causes program termination, the closing of all files, and return to the monitor.

B.8 USEREX

USEREX is a subroutine that allows specification of a routine to which control is passed as part of program termination. This allows disabling of interrupts enabled in non-FORTRAN routines. If these interrupts are not disabled prior to program exit, the integrity of the operating system cannot be assured. The form of the subroutine call is:

```
CALL USEREX (name)
```

where

'name' is the routine to which control is passed and should appear in an EXTERNAL statement somewhere in the program unit. Control is transferred with a JMP instruction after all procedures required for FORTRAN IV program termination have been completed. The transfer of control takes place instead of the normal return to the monitor. Thus, if the user desires to have control passed back to the monitor, the routine specified by USEREX must perform the proper exit procedures.

B.9 RANDU,RAN

The random number generator can be called as a subroutine, RANDU, or as an intrinsic function, RAN. The subroutine call is performed as follows:

```
CALL RANDU (i(1) ,i(2) ,x)
```

where

i(1) and i(2) are previously defined integer variables and x is the real variable name, in which a random number between 0 and 1 is returned. i(1) and i(2) should be initially set to 0. i(1) and i(2) are updated to a new generator base during each call. Resetting i(1) and i(2) to 0 repeats the random number sequence. The values of i(1) and i(2) have a special form; only 0 or saved values of i(1) and i(2) should be stored in these variables.

The random number generator can also be called as a function, as follows:

```
x=RAN(i(1) ,i(2))
```

B.10 SETERR

SETERR allows the user to specify the disposition of certain OTS detected error conditions. Only OTS error diagnostics 1 - 16 should be changed from their default error classification (see Section C.2). If errors 0 or 20 - 69 are changed from the default classification of FATAL, execution continues but in an undetermined state. The form of the call is:

```
CALL SETERR (number,ncount)
```

where

'number' is an integer variable or expression specifying the OTS error number (see Section C.2), and 'ncount' is a LOGICAL*1 variable or expression with one of the following values:

Value	Meaning
0	Ignore all occurrences after logging them on the user terminal.
1	First occurrence of the error will be fatal.
2-127	The nth occurrence of the error will be fatal; the first n-1 occurrences will be logged on the user terminal.
128-255	Ignore all occurrences of the error.

B.11 ERRTST

ERRTST allows the user program to monitor the types of errors detected during program execution. The general form of the call is:

```
CALL ERRTST (ierr,ires)
```

where

ierr is an INTEGER*2 quantity specifying the error number for which the test is to be done.

ires is an INTEGER*2 variable or array element which is to receive the status of the error.

ires=1 if an error has occurred
ires=2 if an error has not occurred

A call to ERRTST will reset the flag governing the specified error condition.

B.12 ERRSNS

ERRSNS allows specifying from zero to two arguments. When ERRSNS is called with zero arguments, previous error data is cleared, thus allowing testing for errors in certain program sections. The general call is:

```
CALL ERRSNS (ires,iunit)
```

where

ires is an INTEGER*2 variable or array element into which the numeric code for the most recent error will be stored. A zero will be stored if there is no error.

iunit is an INTEGER*2 variable or array element into that the logical unit number of input/output errors will be stored, if the most recent error was input/output related.

APPENDIX C

FORTRAN IV ERROR DIAGNOSTICS

C.1 COMPILER ERROR DIAGNOSTICS

The FORTRAN IV Compiler, while reading and processing the FORTRAN source program, can detect syntax errors (or errors in general form) such as unmatched parentheses, illegal characters, unrecognizable key words, and missing or illegal statement parameters.

The error diagnostics are generally clear in specifying the exact nature of the error. In most cases, a check of the general form of the statement in question as described in the PDP-11 FORTRAN Language Reference Manual will help determine the location of the error.

Some of the most common causes of syntax errors, however, are typing mistakes. A typing mistake can sometimes cause the compiler to give very misleading error diagnostics. The user should note, and take care to avoid, the following common typing mistakes:

- Missing commas or parentheses in a complicated expression or FORMAT statement.
- Misspelling of particular instances of variable names. If the compiler does not detect this error (it usually cannot), execution may also be affected.
- An inadvertent line continuation signal on the line following the statement in error.
- If the user terminal does not clearly differentiate between 0 (zero) and the letter O, what appear to be identical spellings of variable names may not appear so to the compiler, and what appears to be a constant expression may not appear so to the compiler.

If any error or warning conditions are detected in a compilation, the following message is printed on the initiating terminal:

```
?FORTRAN-I-[name]Errors:n,Warnings:m
```

where:

[name] is the six-character name of the program unit being compiled. .MAIN. is the default name of the main program if no PROGRAM statement is used. The default name for BLOCK program units is .DATA..

n represents the number of error-class messages (for example, those messages that cause the statement in question to be deleted).

m — represents the number of warning-class messages (for example, those messages indicating conditions that can be ignored or corrected, such as missing END statements or questionable programming practices). Note that some warning conditions will only be detected if the /W switch is specified (see Section C.1.3.).

The next four sections describe the initial phase and secondary phase error diagnostics and the fatal FORTRAN IV Compiler error diagnostics.

The following is an example of a FORTRAN IV program with diagnostics issued by the compiler.

```
FORTRAN IV      V02.5       Thu 01-May-80 00:41:51                PAGE 001

0001          DOUBLE PRECISION DBLE DBLE2
0002          DATA INT/100/ DBLE2/500./
0003          DBLE2 = INT/2 + 5. + DBLE
0004          WRITE,(6,10) DBLE,DBLE
0005    10    FORMAT(1X,2F12.6)
0006    10    STOP
***** M
0007          INTEGER INT
0008          END



FORTRAN IV      Diagnostics for Program Unit .MAIN.

In line 0004, Error:   Syntax error
In line 0008, Warning: Variable 'DBLEDB' name exceeds 6 characters
In line 0009, Warning: Non-standard statement ordering



FORTRAN IV      Storage Map for Program Unit .MAIN.

Local Variables, .PSECT $DATA, Size = 000024 (   10. words)

Name   Type  Offset     Name   Type  Offset     Name   Type  Offset
DBLE   R*4   000020     DBLEDB R*8   000010     DBLE2  R*4   000004
INT    I*2   000002
```

C.1.1 Errors Reported by the Initial Phase of the Compiler

Some of the easily recognizable FORTRAN syntax errors are detected by the initial phase of the compiler. Errors that cause the statement in question to be aborted are tabulated in the ERROR count, whereas those that are correctable by the compiler are counted as WARNINGS.

The error diagnostics are printed after the source statement to which they apply (the L error diagnostic is an exception). The general form of the diagnostic is as follows:

***** c

Where

c is a code letter whose meaning is described below.

INITIAL PHASE ERROR DIAGNOSTICS

Code Letter	Description
B	Columns 1-5 of a continuation line are not blank. Columns 1-5 of a continuation line must be blank except for a possible 'D' in column 1; the columns are ignored (WARNING).
C	Illegal continuation. Comments cannot be continued and the first line of any program unit cannot be a continuation line. If a line consists only of a carriage return/line feed combination, then it is considered to be a blank line. If it has a label field, then it must have a statement field. The line is ignored (WARNING).
E	Missing END statement. An END statement is supplied by the compiler if end-of-file is encountered (WARNING).
H	Hollerith string or quoted literal string is longer than 255 characters or longer than the remainder of the statement; the statement is ignored.
I	Non-FORTRAN character used. The line contains a character that is not in the FORTRAN character set and is not used in a Hollerith string or comment line. The character is ignored (WARNING).
K	Illegal statement label definition. Illegal (non-numeric) character in statement label; the illegal statement label is ignored (WARNING).
L	Line too long to print. There are more than 80 characters (including spaces and tabs) in a line. Note: this diagnostic is issued preceding the line containing too many characters. The line is truncated to 80 characters (WARNING).
M	Multiply defined label. The label is ignored (WARNING).
P	Statement contains unbalanced parentheses. The statement is ignored.
S	Syntax error such as multiple equal signs, etc. The statement is not of the general FORTRAN statement form; the statement is ignored.
U	Statement could not be identified as a legal FORTRAN statement. The statement is ignored.

C.1.2 Errors Reported by Secondary Phases of the Compiler

Those compiler error diagnostics not reported by the initial phase of the compiler appear immediately after the source listing and immediately before the storage map. Since the diagnostics appear after the entire source program has been listed, they must designate the statement to which they apply by using the internal sequence numbers assigned by the compiler.

The general form of the diagnostic is:

```
                    Error:
IN LINE nnnn,               text
                    Warning:
```

where

nnnn is the internal sequence number of the statement in question, and text is a short description of the error.

Below, listed alphabetically, are the error diagnostics. Included with each diagnostic is a brief explanation. Refer to the PDP-11 FORTRAN Language Reference Manual for information to help correct the error.

The notation **** signifies that a particular variable name or statement label appears at that place in the text.

SECONDARY PHASE ERROR DIAGNOSTICS

ACCESS='DIRECT' REQUIRES FORM='UNFORMATTED'
FORM='FORMATTED' has been specified for a direct access file. FORTRAN IV supports only unformatted direct access input/output. Correct the program logic.

ADJUSTABLE DIMENSIONS ILLEGAL FOR ARRAY ****
An adjustable array was not a dummy argument in a subprogram or the adjustable dimensions were not integer dummy arguments in the subprogram. A dimension of one is used. Correct the source program.

ARRAY EXCEEDS MAXIMUM SIZE or
ARRAY **** EXCEEDS MAXIMUM SIZE
The storage required for a single array or for all arrays in total is more than is physically addressable (>32K words). This particular error may reference either the actual statement containing the array in question, or the first statement in the program unit. Correct the statement in error or reduce the space necessary for array storage.

ARRAY **** HAS TOO MANY DIMENSIONS
An array has more than seven dimensions. Correct the program. The legal range for dimensions is one to seven.

**** ATTEMPTS TO EXTEND COMMON BLOCK BACKWARDS
While attempting to EQUIVALENCE arrays in COMMON, an attempt was made to extend COMMON past the recognized beginning of COMMON storage. Correct the program logic.

COMMON BLOCK EXCEEDS MAXIMUM SIZE

An attempt was made to allocate more space to COMMON than is physically addressable (>32K words). Correct the statement in error.

CONSTANT IN FORMAT STATEMENT NOT IN RANGE

An integer constant in a FORMAT statement was not in the proper range. Check that all integer constants are within the legal range (1 to 255).

DANGLING OPERATOR

An operator (+,-,*,/, etc.) is missing an operand. Example: I=J+. Correct the statement in error.

DEFECTIVE DOTTED KEYWORD

A dotted relational operator was not recognized or there is a possible misuse of a decimal point. Check the format for relational operators; correct the statement in error.

DEFINE FILE MODE MUST BE 'U'

The third argument inside parentheses in a DEFINE FILE statement is not 'U' (unformatted). Correct the mode specification.

DO TERMINATOR **** PRECEDES DO STATEMENT

The statement specified as the terminator of a DO loop did not appear after the DO statement. Correct the program logic.

EXPECTING LEFT PARENTHESIS AFTER ****

An array name or function name reference is not followed by a left parenthesis. Correct the statement so that the left parenthesis is included.

EXPECTING LEFT PARENTHESIS AFTER SUBPROGRAM NAME

A SUBROUTINE or FUNCTION name occurs without an argument list specification. Check for a typographical error, or the use of the same name for a local variable and a subprogram.

EXTRA CHARACTERS AT END OF STATEMENT

All the necessary information for a syntactically correct FORTRAN statement has been found on this line, but more information exists. Check that a comma is not missing from the line or that an unintentional continuation signal does not appear on the next line.

FLOATING CONSTANT NOT IN RANGE

A floating constant in an expression is too close to zero to be represented in the internal format. Use zero if possible.

ILLEGAL ADJACENT OPERATOR

Two operators (*,/, logical operators, etc.) are illegally placed next to each other. Example: I/*J. Correct the statement in error.

ILLEGAL CHARACTERS IN EXPRESSION

An illegal character has been found in an expression. Check for a typographical error in the statement.

ILLEGAL DO TERMINATOR ORDERING AT LABEL ****
DO loops are nested improperly. Verify that the range of each DO loop lies completely within the range of the next outer loop.

ILLEGAL DO TERMINATOR STATEMENT ****
A DO statement terminator was not valid. Verify that the DO statement terminator is not a GOTO, arithmetic IF, RETURN, another DO statement, or logical IF containing one of these statements.

ILLEGAL ELEMENT IN I/O LIST
An item, expression, or implied DO specifier in an I/O list is of illegal syntax. Correct the I/O list.

ILLEGAL ENCODE/DECODE FORMAT SPECIFIER
The format specification (second argument inside parentheses) in an ENCODE/DECODE statement is not a FORMAT statement label or array name. Correct the FORMAT specification.

ILLEGAL ENCODE/DECODE LENGTH EXPRESSION
The length specification (first argument inside parentheses) in an ENCODE or DECODE statement is not an integer expression. Correct the length expression.

ILLEGAL ENCODE/DECODE TARGET
The third argument inside parentheses in an ENCODE or DECODE statement is not the name of an array, array element, or variable. Correct the target specification.

ILLEGAL INITIAL VALUE EXPRESSION IN DO STATEMENT
A valid integer expression does not follow the equals sign in a DO statement. Correct the initial value expression.

ILLEGAL STATEMENT IN BLOCK DATA
An illegal statement was found in a BLOCK DATA subprogram. Verify that a FORMAT or executable statement does not occur in a BLOCK DATA subprogram.

ILLEGAL STATEMENT ON LOGICAL IF
The statement contained in a logical IF was not valid. Verify that the statement is not another logical IF or DO statement.

ILLEGAL SUBSCRIPTS OR SUBPROGRAM ARGUMENT
An illegal element occurred within a subscript list or argument list to a subprogram. Correct the erroneous statement.

ILLEGAL TYPE FOR OPERATOR
An illegal variable type has been used with an exponentiation or logical operator. Check that the variable type is valid for the operation in question.

ILLEGAL USAGE OF OR MISSING LEFT PARENTHESIS
A left parenthesis was required but not found, or a variable reference or constant is illegally followed by a left parenthesis. Correct the format of the statement in error.

INTEGER OVERFLOW

An integer constant or expression value is outside the range -32767 to +32767. Correct the value of the integer constant or expression so that it falls within the legal range (-32767 to +32767).

INVALID COMPLEX CONSTANT

A complex constant has been improperly formed. Correct the statement in error.

INVALID DIMENSIONS FOR ARRAY ****

An attempt was made, while dimensioning an array, to explicitly specify zero as one of the dimensions. Verify that zero is not used as a dimension.

INVALID END= OR ERR= KEYWORD

The END= or ERR= specification in an input/output statement is incorrectly formatted. Check for a typographical error in the statement.

INVALID EQUIVALENCE

An illegal EQUIVALENCE, or EQUIVALENCE that is contradictory to a previous EQUIVALENCE, was encountered. Correct the program logic.

INVALID FORMAT SPECIFIER

A format specifier was illegally used. Correct the statement so that the format specifier is the label of a FORMAT statement or an array name.

INVALID IMPLICIT RANGE SPECIFIER

An illegal implicit range specifier, (i.e., non-alphabetic specifier, or specifier range in reverse alphabetic order) was encountered. Verify that the implicit range specifier indicates alphabetic characters in alphabetic order.

INVALID LOGICAL UNIT

A logical unit reference was incorrect. Correct the logical unit reference so that it is an integer variable or constant in the range 1 to 99.

INVALID OCTAL CONSTANT

An octal constant is too large or contains a digit other than 0-7. Correct the constant so that it contains only legal digits that fall within the octal range 0 to 177777.

INVALID OPTIONAL LENGTH SPECIFIER

A data type declaration optional length specifier is illegal. For example, REAL*4 and REAL*8 are legal, but REAL*6 is not. Correct the statement so that it contains only a valid data type declaration length.

INVALID RADIX-50 CONSTANT

An illegal character was detected in a RADIX-50 constant. Verify that only characters from the RADIX-50 character set are used in a RADIX-50 constant.

INVALID STATEMENT LABEL REFERENCE

Reference has been made to a statement number that is of illegal construction. For example, GOTO 999999 is illegal since the statement number is too long. Check that the statement number consists of one to five

decimal digits placed in the first five columns of a statement's initial line and that it does not contain only zeroes.

INVALID PROGRAM NAME
A name used in a CALL statement or function reference is not valid. For example, use of an array name in a CALL statement routine name reference is illegal. Verify that the name specified in the statement is spelled correctly.

INVALID TARGET FOR ASSIGNMENT
The left side of an arithmetic assignment statement is not a variable name or array element reference. Correct the statement in error.

INVALID TYPE SPECIFIER
An unrecognizable data type was used. Verify that the data type indicated is valid.

INVALID USAGE OF SUBROUTINE OR FUNCTION NAME
A function name appeared in a DIMENSION, COMMON, DATA, or EQUIVALENCE or data type declaration statement. Correct the statement in error.

INVALID VARIABLE NAME
A variable name contains an illegal character, is missing, or does not begin with an alphabetic character. Correct the statement in error.

LABEL ON DECLARATIVE STATEMENT
A label was found on a declarative statement. Correct the program so that declarative statements do not have labels.

MISSING ASSIGNMENT OPERATOR
The first operator seen in an arithmetic assignment statement was not an equal sign (=). For example, I+J=K. Correct the arithmetic assignment statement in error.

MISSING COMMA
The comma delimiter was expected but not found. Correct the format of the statement in error.

MODE OF EXPRESSION MUST BE INTEGER
An integer variable or expression is required, or, for example, in the initial, terminal, and incremental parameter of a DO statement.

MISSING COMMA IN OPEN OR CLOSE KEYWORD LIST
Two options in an OPEN/CLOSE keyword list are not separated by a comma. Check for a typographical error in the statement.

MISSING DELIMITER IN EXPRESSION
Two operands have been placed next to each other in an expression with no operator between them. Correct the statement in error.

MISSING EXPRESSION
A required expression (for example, the limit expression in a DO statement) was omitted. Correct the syntax of the statement.

MISSING LABEL
A statement label was expected but not found. For example, ASSIGN J TO I is illegal; a valid statement label reference should precede 'TO' but does not. Verify that the reference preceding 'TO' is a valid statement label of an executable statement in the same program unit as the ASSIGN statement.

MISSING LABEL LIST AFTER COMMA
In an assigned GOTO statement, the integer variable was followed by a comma but no list was found. Check for a typographical error in the statement.

MISSING LEFT PARENTHESIS AFTER OPEN OR CLOSE
An OPEN or CLOSE statement does not have a left parenthesis preceding the keyword list. Check for a typographical error in the statement.

MISSING OPERATOR AFTER EXPRESSION
An expression was not terminated by a comma, right parenthesis, or other operator. Check for a typographical error in the statement.

MISSING QUOTATION MARK
In a FIND statement, the logical unit number and record number were not separated by a single quotation mark. Correct the statement in error.

MISSING RIGHT PARENTHESIS
A right parenthesis was expected but not found. For example, READ(5,100,) is illegal; the first nonblank character after the format reference should be a right parenthesis but is not. Correct the format of the statement in error.

MISSING 'TO' IN ASSIGN STATEMENT
The keyword 'TO' does not follow the label specification in an ASSIGN statement. Check for a typographical error in the statement.

MISSING VALUE FOR KEYWORD IN OPEN OR CLOSE STATEMENT
A keyword requiring a value was specified without a value. Correct the syntax of the statement.

MISSING VARIABLE
A variable was expected but not found. For example, ASSIGN 100 TO 1 is illegal; a variable name should follow the 'TO' but does not. Verify that the reference following 'TO' is a valid integer variable name.

MISSING VARIABLE OR CONSTANT
An operand (variable or constant) was expected but a delimiter (comma, parenthesis, etc.) was found. For example, WRITE() is illegal; a unit number should follow the open parenthesis, but a delimiter (close parenthesis) is encountered instead. Correct the format of the statement in error.

MODE OF EXPRESSION MUST BE INTEGER
An integer variable or expression is required, as, for example, in the initial, terminal, and incremental parameters of a DO statement.

MODES OF VARIABLE **** AND DATA ITEM DIFFER
The data type of each variable and its associated data list item must agree in a DATA statement. Correct the format of the items in the DATA statement.

MULTIPLE DECLARATION FOR VARIABLE ****
A variable appeared in more than one data type declaration statement or dimensioning statement. Subsequent declarations are ignored. Correct the program logic.

MULTIPLE DECLARATION OF OPEN OR CLOSE KEYWORD
A keyword has been specified more than once in a single OPEN or CLOSE statement. Remove the incorrect or duplicate reference to the keyword.

OPEN OR CLOSE KEYWORD VALUE MUST BE QUOTED STRING
A keyword that requires a quoted-string value was given an expression value. Correct the syntax of the statement.

OPEN OR CLOSE STATEMENT REQUIRES UNIT= SPECIFIER
No UNIT= specification is present in the OPEN or CLOSE statement in question to select the desired logical unit. Add the UNIT= specification to the s^Dtement.

PARENTHESES NESTED TOO DEEPLY
Group repeats in a FORMAT statement have been nested too deeply. Limit group repeats to eight levels of nesting.

PROGRAM OR BLOCK DATA STATEMENT MUST BE FIRST
If either a program name statement or block data name statement is used, it should always be the first statement.

P-SCALE FACTOR NOT IN RANGE -127 TO +127
P-scale factors were not in the range -127 to +127. Correct the statement in error.

REFERENCE TO INCORRECT TYPE OF LABEL ****
A statement label reference that should be a label on a FORMAT statement is not such a label, or a statement label reference that should be a label on an executable statement is not such a label. Correct the program logic.

REFERENCE TO UNDEFINED STATEMENT LABEL
A reference has been made to a statement label that has not been defined anywhere in the program unit. Correct the program logic.

STATEMENT MUST BE UNLABELED
A DATA, SUBROUTINE, FUNCTION, BLOCK DATA, arithmetic statement function definition, or declarative statement was labeled. Correct the statement in error.

STATEMENT TOO COMPLEX

An arithmetic statement function has more than ten dummy arguments or the statement is too long to compile. Verify that the number of dummy arguments in an arithmetic statement does not exceed ten; break long statements into two or more smaller statements.

SUBROUTINE OR FUNCTION STATEMENT MUST BE FIRST

A SUBROUTINE, FUNCTION, or BLOCK DATA statement is not the first statement in a program unit. Ensure that, if present, these statements appear first in a program unit.

SUBSCRIPT OF ARRAY **** NOT IN RANGE

Array subscripts that are constants or constant expressions are found to be outside the bounds of the array's dimensions. The operation in question is aborted. Correct the program.

SYNTAX ERROR

The general form of the statement was not formatted correctly. Check the general format of the statement in error and correct the program.

SYNTAX ERROR IN INTEGER OR FLOATING CONSTANT

An integer or floating constant has been incorrectly formed. For example, 1.23.4 is an illegal floating constant because it contains two decimal points. Correct the format of the integer or floating constant in question.

SYNTAX ERROR IN LABEL LIST

The list of labels for an assigned or computed GOTO statement is improperly formatted, or contains a reference to an entity that is not the label on an executable statement. Correct the format of the list.

TARGET MUST BE ARRAY

An array element was referenced in an ENCORE or DECODE statement without having been previously dimensioned.

UNARY OPERATOR HAS TOO MANY OPERANDS

Two operands have been specified for an operator (such as .NOT.) that accepts only one operand. Check for a typographical error in the statement.

UNLABELED FORMAT STATEMENT

A FORMAT statement was not labeled. Correct the FORMAT statement in error by assigning it the proper label.

UNRECOGNIZED KEYWORD IN OPEN OR CLOSE STATEMENT

The OPEN or CLOSE statement in question contains a keyword that is not recognized by the compiler. Check for a misspelling or typographical error in the keywords used in the statement.

UNRECOGNIZED VALUE FOR OPEN OR CLOSE KEYWORD

A keyword which requires a quoted-string value was specified with an unrecognized value string. For example, DISPOSE = 'SURE'. Specify a valid value for the keyword.

USAGE OF VARIABLE **** INVALID
An attempt was made to EXTERNAL a common variable, an array variable, or a dummy argument. Or an attempt was made to place in COMMON a dummy argument or external name. Correct the program logic.

VALUE OF CONSTANT NOT IN RANGE
An integer constant in the designated source program line exceeds the maximum unsigned value (65535). This error is also printed if an invalid dimension is specified for an array or if the exponent of a floating point constant is too large. Correct the statement in error.

VARIABLE **** INVALID IN ADJUSTABLE DIMENSION
A variable used as an adjustable dimension was not an integer dummy argument in the subprogram unit. Correct the program.

WRONG NUMBER OF OPERANDS FOR BINARY OPERATOR
An operator that requires two operands was specified with only one operand. Example: I=*J. Check for a typographical error in the statement.

WRONG NUMBER OF SUBSCRIPTS FOR ARRAY ****
An array reference does not have the same number of subscripts as specified when the array was dimensioned. Correct the statement in error.

C.1.3 Warning Diagnostics

Warning diagnostics report conditions that are not true error conditions, but that can be potentially dangerous at execution time, or can present compatibility problems with FORTRAN compilers running on other DEC operating systems. The warning diagnostics are normally suppressed, but can be enabled by use of the /WARNINGS (/W) compiler option. The form and placement of the warning diagnostics are the same as those for the secondary phase error diagnostics (see Section C.1.2). A listing of the warning diagnostics follows.

LOOP ENTRY AT LABEL ****
A transfer of control occurs from outside the containing DO loop to the label indicated. This may indicate a programming error (if the loop does not have extended range).

POSSIBLE MODIFICATION OF ****
The indicated variable, which is used as a control parameter of a DO loop, may be modified within the body of that loop.

NON-STANDARD STATEMENT ORDERING
Although the FORTRAN IV Compiler has less restrictive statement-ordering requirements than those outlined in Chapter 7 of the PDP-11 FORTRAN Language Reference Manual, non-adherence to the stricter requirements may cause error conditions on other FORTRAN compilers. See Section 3.1 of this document.

VARIABLE **** IS NOT WORD ALIGNED
Placing a non-LOGICAL*1 variable or array after a LOGICAL*1 variable or array in COMMON, or equivalencing

non-LOGICAL*1 variables or arrays to LOGICAL*1 variables or arrays can cause this condition. An attempt to reference the variable at run time will cause an error condition.

VARIABLE **** NAME EXCEEDS SIX CHARACTERS
A variable name of more than six characters was specified. The first six characters were used as the true variable name. Other FORTRAN compilers may treat this as an error condition. See Section 3.2 of this document.

C.1.4 Fatal Compiler Error Diagnostics

Listed below are the fatal compiler error diagnostics. These diagnostics, which are sent directly to the initiating terminal, report hardware error conditions, conditions that may require rewriting of the source program, and compiler errors that may require attention from your Software Support Representative.

?FORTRAN-F-CODE GENERATION STACK OVERFLOW
A statement is too complex to process. Simplify complex statements.

?FORTRAN-F-COMPILER FATAL ERROR, ANALYSIS FOLLOWS
Some type of unexpected error was encountered by FORTRAN. Please send the console listing with a copy of your program (on some machine-readable medium) with an SPR report.

?FORTRAN-F-CONSTANT SUBSCRIPT STACK OVERFLOW
Too many constant subscripts have been employed in a statement. Simplify the statement.

?FORTRAN-F-DEVICE FULL
There is insufficient room available on an output device specified to create the object or listing files required. Make more space available on the device by deleting unnecessary files and/or using the SQUEEZE command, or by redirecting the object on listing files to another device.

?FORTRAN-F-DYNAMIC MEMORY OVERFLOW
The program unit currently being compiled cannot be processed in the available memory space. Break the program unit in question into smaller subprograms, or recompile on a larger machine.

?FORTRAN-F-ERROR READING SOURCE FILE
An unrecoverable error occurred while the compiler was attempting to read a source program input file. Correct the hardware problem and recompile.

?FORTRAN-F-ERROR WRITING LISTING FILE
An unrecoverable error occurred while the compiler was attempting to write the listing output file. Make sure that the output device is write-enabled, and that sufficient free space exists on the device for the output file. Recompile the program.

?FORTRAN-F-ERROR WRITING OBJECT FILE
An unrecoverable error occurred while the compiler was attempting to write the object program output file. Make sure that the output device is write-enabled, and that sufficient free space exists on the device for the output file. Recompile the program.

?FORTRAN-F-FILE NOT FOUND
An input file specified in the command string was not found. Correct the command string to refer to an existing file.

?FORTRAN-F-HELP FILE NOT FOUND
The FORTRAN IV help file, SY:FORTRA.HLP, was not present on the system device when the help switch was given to the compiler. No help information is available. Replace the file from the FORTRAN distribution medium if help information is required.

?FORTRAN-F-ILLEGAL VALUE FOR /x SWITCH
An illegal value has been specified for a compiler command string switch. Refer to Section 1.2.2 for compiler option information.

?FORTRAN-F-ILLEGAL COMMAND
The command string presented to the compiler was illegal in format. Correct the command string.

?FORTRAN-F-ILLEGAL DEVICE
A device specification in a compiler command string was illegal. Correct the command string.

?FORTRAN-F-OPTIMIZER STACK OVERFLOW
A statement is too complex to process, or too many common subexpressions occurred in one basic block of the source program. Simplify complex statements.

?FORTRAN-F-SUBEXPRESSION STACK OVERFLOW
An attempt was made to compile a statement that could overflow the runtime stack at execution time. Simplify complex statements.

?FORTRAN-F-UNKNOWN SWITCH-/x
An illegal switch has been specified in the compiler command string. Refer to Section 1.2.2 for compiler option information.

C.2 OBJECT-TIME SYSTEM ERROR DIAGNOSTICS

The Object-Time System detects certain I/O, arithmetic, and system failure error conditions and reports them on the user terminal. These error diagnostics are printed in either a long or short form.

The short form of the message appears as:

```
?ERR nn
```

where nn is a decimal error identification number.

The long form of the message appears as:

?ERR nn text

where nn is a decimal error identification number and text is a short error description.

For RT-11, the default message form is long. The short message error module can be linked to the program by using the /I linker switch. The global named $SHORT should be included from the FORTRAN library. For RSTS/E, whether the message format is long or short depends upon how the OTS Library was built.

There are four classes of OTS error conditions. Each error condition is assigned to one of these classes. An error condition classification for the error codes 1-16 can be changed by using the system subroutine SETERR. (See Section B.10). Error codes 0 and 20-69 should not be changed from their FATAL classification or indeterminable results will occur. The classifications are:

IGNORE	the error is detected but no error message is sent to the terminal. Execution continues.
WARNING	the error message is sent to the terminal and execution continues.
FATAL	the error message is sent to the terminal and execution is terminated.
COUNT:n	the error message is sent to the terminal and execution continues until the nth occurrence of the error, at which time the error will be treated as FATAL.

If a program is terminated by a fatal error condition, active files may not be closed. Under RT-11, when control is returned to the Monitor, a CLOSE command can be given to close all active files, although some of the output to these active files may have been lost.

The OTS error diagnostics are listed below, along with the error type and a brief explanation, where necessary.

Error number	Error type	Message
0	FATAL	NON-FORTRAN ERROR CALL This message indicates an error condition (not internal to the FORTRAN run-time system) that may have been caused by one of four situations: (RT-11 only) 1. A foreground job using SYSLIB completion routines was not allocated enough space (using the FRUN /N option) for the initial call to a completion routine. Check Chapter 4 (SYSTEM SUBROUTINE LIBRARY) of the RT-11 Advanced Programmers Guide for the formula used to allocate more space.

(continued on next page)

Error number	Error type	Message
		(RT-11 only) 2. There was not sufficient memory for the background job. Make more memory available by unloading unnecessary handlers, deleting unwanted files, compressing the device. (RT-11 only) 3. Under the single-job monitor, a SYSLIB completion routine interrupted another completion routine. Use the FB Monitor to allow more than one active completion routine. 4. An assembly language module linked with a FORTRAN program issued a TRAP instruction with an error code that was not recognized by the FORTRAN error handler. Check the program logic.
1	FATAL	INTEGER OVERFLOW During an integer multiplication, division, or exponentiation operation, the value of the result exceeded 32767 in magnitude. Correct the program logic.
2	FATAL	INTEGER ZERO DIVIDE During an integer mode arithmetic operation, an attempt was made to divide by zero. Correct the program logic.
3	FATAL	COMPILER GENERATED ERROR An attempt was made to execute a FORTRAN statement in which the compiler had previously detected errors. Consult the program listing generated by the compiler (if one was requested) and correct the program for the errors detected at compile time.
4	WARNING	COMPUTED GOTO OUT OF RANGE The value of the integer variable or expression in a computed GOTO statement was less than one or greater than the number of statement label references in the list. Control is passed to the next executable statement. Examine the source program and correct the program logic.
5	COUNT:3	INPUT CONVERSION ERROR During a formatted input operation, an illegal character was detected in an input field.

(continued on next page)

Error number	Error type	Message
		A value of zero is returned. Examine the input data and correct the invalid record.
6	IGNORE	OUTPUT CONVERSION ERROR During a formatted output operation, the value of a particular number could not be output in the specified field length without loss of significant digits. The field is filled with asterisks ('*'). Correct the FORMAT statement to allow a greater field length.
10	COUNT:3	FLOATING OVERFLOW During an arithmetic operation, the absolute value of a floating-point expression exceeded the largest representable real number. A value of zero is returned. Correct the program logic.
11	IGNORE	FLOATING UNDERFLOW During an arithmetic operation, the absolute value of a floating-point expression became less than the smallest representable real number. The real number is replaced with a value of zero. Correct the program logic.
12	FATAL	FLOATING ZERO DIVIDE During a REAL mode arithmetic operation an attempt was made to divide by zero. The result of the operation is set to zero. Correct the program logic.
13	COUNT:3	SQRT OF NEGATIVE NUMBER An attempt was made to take the square root of a negative number. The result is replaced by zero. Correct the program logic.
14	FATAL	UNDEFINED EXPONENTIATION OPERATION An attempt was made to perform an illegal exponentiation operation. (For example, -3.**.5 is illegal because the result would be an imaginary number.) The result of the operation is set to zero. Correct the program logic.
15	FATAL	LOG OF ZERO OR NEGATIVE NUMBER An attempt was made to take the logarithm of a negative number or zero. The result of the operation is set to zero. Correct the program logic.

(continued on next page)

Error number	Error type	Message
16	FATAL	WRONG NUMBER OF ARGUMENTS One of the FORTRAN Library functions, or one of the system subroutines that checks for such an occurrence, was called with an improper number of arguments. Check the format of the particular library function or system subroutine call, and correct the call.

The following error diagnostics should not be changed from the FATAL classification by use of the system subroutine SETERR:

20	FATAL	INVALID LOGICAL UNIT NUMBER An illegal logical unit number was specified in an I/O statement. A logical unit number must be an integer within the range 1 to 99. Correct the statement in error.
21	FATAL	OUT OF AVAILABLE LOGICAL UNITS An attempt was made to have too many logical units simultaneously open for I/O. The maximum number of active logical units is six by default. To increase the maximum, recompile the main program using the /UNITS (/N) option to specify a larger number of available channels.
22	FATAL	INPUT RECORD TOO LONG During an input operation, a record was encountered that was longer than the maximum record length. The default maximum record length is 136 (decimal) bytes. To increase the maximum, recompile the main program using the /RECORDS (/R) option to specify a larger run-time record buffer (the legal range is 4 to 4095).
23	FATAL	HARDWARE I/O ERROR A hardware error was detected during an I/O operation. Check the volume for an off-line or write-locked condition, and retry the operation. Try another unit or drive if possible, or use another device.
24	FATAL	ATTEMPT TO READ/WRITE PAST END OF FILE During a sequential READ operation, an attempt was made to read beyond the last record of the file. During a random access READ, this message indicates that an attempt was made to reference a record number that was not within the bounds of the file.

(continued on next page)

Error number	Error type	Message
		Use the "END=" parameter to detect this condition, or correct the program logic so that no request is made for a record outside the bounds of the file. During a WRITE operation, this message indicates that the space available for the file is insufficient. (RT-11 action) Try to make more file space available by deleting unnecessary files and compressing the device, or by using another device. (RSTS/E action) This condition is equivalent to the "NO ROOM FOR USER ON DEVICE" system error. Make more space available by deleting files from the current account, or use another device.
25	FATAL	ATTEMPT TO READ AFTER WRITE An attempt was made to read after writing on a sequential file located on a file-structured device. A write operation must be followed by a REWIND or BACKSPACE before a read operation can be performed. Correct the program logic.
26	FATAL	RECURSIVE I/O NOT ALLOWED An expression in the I/O list of a WRITE statement caused initiation of another READ or WRITE operation. (This can happen if a FUNCTION that performs I/O is referenced within an expression in an I/O list.) Correct the program logic.
27	FATAL	ATTEMPT TO USE DEVICE NOT IN SYSTEM An attempt was made to access a device that was not legal for the system in use. Use the system ASSIGN command to create the required logical device name, or change the statement in error.
28	FATAL	OPEN FAILED FOR FILE The file specified was not found, there was no room on the device, or there was an attempt to create a file which already exists as a protected file. Verify that the file exists as specified. Delete unnecessary files from the device, or use another device.
29	FATAL	NO ROOM FOR DEVICE HANDLER (RT-11 only) There was not enough free memory left to accommodate a specific device handler.

(continued on next page)

Error number	Error type	Message
		Move the file to the system device or to a device whose handler is resident. Make more memory available by unloading unnecessary handlers, unloading the foreground job, using the single-job Monitor, or SET USR SWAP, if possible.
30	FATAL	NO ROOM FOR BUFFERS There was not enough free memory left to set up required I/O buffers. (RT-11 only) Reduce the number of logical units that are open simultaneously at the time of the error. If using double buffering or if another file is currently open, use single buffering. Make more memory available by unloading unnecessary handlers, unloading the foreground job, using the single-job Monitor, or SET USR SWAP, if possible. (RSTS/E only) Increase the space available for buffering by specifying the appropriate value for the /K switch to LINK, or reduce the number of logical units that are open simultaneously at the time of the error.
31	FATAL	NO AVAILABLE I/O CHANNEL More than the maximum number of channels available to the FORTRAN IV run-time system (15 for RT-11; 14 for RSTS/E, exclusive of the terminal) were requested to be simultaneously opened for I/O. Close any logical units previously opened that need not be open at this time.
32	FATAL	FMTD-UNFMTD-RANDOM I/O TO SAME FILE An attempt was made to perform any combination of formatted, unformatted, or random access I/O to the same file. Correct the program logic.
33	FATAL	ATTEMPT TO READ PAST END OF RECORD An attempt was made to read a larger record than actually existed in a file. Check the construction of the data file; correct the program logic.
34	FATAL	UNFMTD I/O TO TT OR LP An attempt was made to perform an unformatted write operation on the terminal or line printer. Assign the logical unit in question to the appropriate device, using the ASSIGN system command, the OPEN statement, the ASSIGN or OPEN FORTRAN library routine, or (RT-11 only) the IASIGN SYSLIB routine.

(continued on next page)

Error number	Error type	Message
35	FATAL	ATTEMPT TO OUTPUT TO READ ONLY FILE An attempt was made to write on a file designated as read only. Check the OPEN statement, the ASSIGN or OPEN (RSTS/E only) system routine, or IASIGN SYSLIB function (RT-11 only) to ensure that the correct arguments were used. Check for a possible programming error.
36	FATAL	BAD FILE SPECIFICATION STRING The Hollerith or literal string specifying the device/file name OPEN statement, in the CALL ASSIGN or CALL OPEN system subroutine, could not be interpreted. Check the format of the OPEN statement, CALL ASSIGN, or CALL OPEN statement.
37	FATAL	RANDOM ACCESS READ/WRITE BEFORE DEFINE FILE A random access read or write operation was attempted before a DEFINE FILE was performed. Correct the program so that the DEFINE FILE operation is executed before any random-access read or write operation.
38	FATAL	RANDOM I/O NOT ALLOWED ON TT OR LP Random access I/O was illegally attempted on the terminal or line printer. Assign the logical unit in question to the appropriate device, using the ASSIGN keyboard monitor command, OPEN statement, the ASSIGN or OPEN FORTRAN library routine, or (RT-11 only) the IASIGN SYSLIB routine.
39	FATAL	RECORD LARGER THAN RECORD SIZE IN DEFINE FILE A record was encountered that was larger than that specified in the DEFINE FILE statement for a random access file. Shorten the I/O list or redefine the file specifying larger records.
40	FATAL	REQUEST FOR A BLOCK LARGER THAN 65535 An attempt was made to reference an absolute disk block address greater than 65535. Correct the program logic.
41	FATAL	DEFINE FILE ATTEMPTED ON AN OPEN UNIT A file was open on a unit and another DEFINE FILE was attempted on that unit. Close the open file using the CLOSE statement before attempting another DEFINE FILE.

(continued on next page)

Error number	Error type	Message
42	FATAL	MEMORY OVERFLOW COMPILING OBJECT TIME FORMAT The OTS ran out of free memory while scanning an array format generated at run time. (RT-11 only) Use a FORMAT statement specification at compile time rather than object-time formatting, or make more memory available by unloading unnecessary handlers, unloading the foreground job if possible, using the single-job Monitor, or SET USR SWAP, if possible. (RSTS/E only) Use a FORMAT statement specification at compile time rather than object-time formatting, or allocate more space by using the /K switch to LINK.
43	FATAL	SYNTAX ERROR IN OBJECT TIME FORMAT A syntax error was encountered while the OTS was scanning an array format generated at run time. Correct the programming error.
44	FATAL	2ND RECORD REQUEST IN ENCODE/DECODE An attempt was made to use ENCODE and DECODE on more than one record. Correct the FORMAT statement associated with the ENCODE or DECODE so that it specifies only one record.
45	FATAL	INCOMPATIBLE VARIABLE AND FORMAT TYPES An attempt was made to output a real variable with an integer field descriptor or an integer variable with a real field descriptor. Correct the FORMAT statement associated with the READ or WRITE, ENCODE or DECODE.
46	FATAL	INFINITE FORMAT LOOP The format associated with an I/O statement, which includes an I/O list, had no field descriptors to use in transferring those variables. Correct the FORMAT statement in error.
47	FATAL	ATTEMPT TO STORE OUTSIDE PARTITION (RT-11 only) In an attempt to store data into a subscripted variable, the address calculated for the array element in question did not lie within the section of memory allocated to the job. The subscript in question was out of bounds. (This message is issued only when bounds checking modules have been installed in FORLIB and threaded code is selected at compile time.) Correct the program logic.

(continued on next page)

Error number	Error type	Message
48	FATAL	UNIT ALREADY OPEN An attempt was made to perform an illegal operation on an open file.
49	FATAL	ENDFILE ON RANDOM FILE An ENDFILE statement contains a unit number of a file that is open as a random access file.
50	FATAL	KEYWORD VALUE ERROR IN OPEN STATEMENT A numeric value specified for a keyword in the OPEN statement is not within the accepted range. Check the expression in the OPEN statement, and modify to yield a valid value.
51	FATAL	INCONSISTENT OPEN/CLOSE STATEMENT SPECIFICATIONS The specifications in an OPEN or subsequent CLOSE statement have indicated one or more of the following: • A 'NEW' or 'SCRATCH' file that is 'READONLY'. • 'APPEND' to a 'NEW', 'SCRATCH', or 'READONLY' file. • 'SAVE' or 'PRINT' of a 'SCRATCH' file. • 'DELETE' or 'PRINT' of a 'READONLY' file. Correct the OPEN or CLOSE statement to remove the conflict.
52	WARNING	ATTEMPT TO DELETE A PROTECTED FILE (RT-11 only). An attempt was made through a DISP= 'DELETE' option in an OPEN or CLOSE statement to delete a protected file. Either unprotect the file or correct the conflict in the OPEN and/or CLOSE statement.
53	WARNING	LIST DIRECTED I/O SYNTAX ERROR The repeat count of the input record has the wrong type or value. The repeat count must be a positive non-zero integer.
59	WARNING	USR NOT LOCKED (RT-11 only) This message is issued when the FORTRAN program is started. If the program was running in the foreground, the /NOSWAP option was used during compilation, and the USR was swapping (for example, a SET USR NOSWAP command has not been done). Reexamine the intent of the /NOSWAP option at compile time and either compile without /NOSWAP or issue a SET USR NOSWAP command.

(continued on next page)

Error number	Error type	Message
60	FATAL	STACK OVERFLOWED The hardware stack overflowed. More stack space may be required for subprogram calls and opening of file. Proper traceback is impaired. This message occurs in the background only. Allocate additional space by using the /BOTTOM (/B) option at link time. Check for a programming error.
61	FATAL	ILLEGAL MEMORY REFERENCE Some type of bus error occurred, most probably an illegal memory address reference. If an assembly language routine was called, check for a coding error in the routine. Otherwise, insure that the correct FORTRAN library was called.
62	FATAL	FORTRAN START FAIL The program was loaded into memory but there was not enough free memory remaining for the OTS to initialize work space and buffers. (RT-11 only) If running a background job, make more memory available by unloading unnecessary handlers, using the single-job Monitor. If running a foreground job, specify a larger value using the FRUN /N option. Refer to Chapter 4 (SYSTEM SUBROUTINE LIBRARY) of the RT-11 Advanced Programmers Guide for the correct formula. (RSTS/E only) Allocate more space by using the /K switch to LINK.
63	FATAL	ILLEGAL INSTRUCTION The program attempted to execute an illegal instruction (for example, floating-point arithmetic instruction on a machine with no floating-point hardware). If an assembly language routine was called, check for a coding error in the routine. Otherwise, ensure that the correct FORTRAN library was called.
64	FATAL	VIRTUAL ARRAY INITIALIZATION FAILURE • The total storage requirements for VIRTUAL arrays in the program exceeds the currently available memory on the system. • Another job is currently using VIRTUAL support under the FB or SJ monitor. • PLAS VIRTUAL array support is attempted under the FB or SJ monitor.

(continued on next page)

Error number	Error type	Message
		• Any non-PLAS VIRTUAL array support is attempted under XM monitor. • PLAS support is attempted without EIS hardware. • VIRTUAL array support is attempted without OTS Library VIRTUAL support. If another job is currently executing (under XM monitor with PLAS VIRTUAL support), make sure that the total VIRTUAL storage demands for both programs are satisfied by the available memory on the machine. Reduce VIRTUAL storage requirements by decreasing the size of VIRTUAL arrays declared in the program.
65	FATAL	VIRTUAL ARRAY MAPPING ERROR An attempt has been made to reference outside the bounds of the extended memory region allocated to VIRTUAL arrays in this program, probably caused by a subscript out of bounds. Verify that the subscripts of the VIRTUAL arrays referenced in the statement indicated are within the declared bounds.
66	FATAL	UNSUPPORTED OPEN/CLOSE KEYWORD OR OPTION A keyword or keyword value in the OPEN or CLOSE statement indicated is invalid under this particular operating system. Refer to Section 3.1.1 for system-dependent restrictions.
67	WARNING	UNSUPPORTED OPEN/CLOSE KEYWORD OR OPTION A keyword or keyword value in the OPEN or CLOSE statement indicated is not meaningful in the current operating system environment. The presence of the keyword or option is ignored.
68	FATAL	DIRECT ACCESS RECORD SIZE ERROR The size of a direct access record exceeds 32767 double words. Correct the program logic.
69	FATAL	Cannot send to QUEMAN (RSTS/E only). An error occurred when closing a file with attribute DISPOSE='PRINT'.

APPENDIX D

COMPATIBILITY WITH OTHER PDP-11 LANGUAGE PROCESSORS

FORTRAN IV is an implementation of FORTRAN for the PDP-11, available under RT-11, RSTS/E, RSX-11M, RSX-11M-PLUS, VAX/VMS under AME, and IAS. It has been designed to be upward compatible with FORTRAN IV PLUS (F4P) and compatible with the earlier FORTRAN (FTN) on DOS-11 and RSX-11D V4A. However, some differences exist as a result of:

- correcting deficiencies in FTN FORTRAN.
- language specification decisions necessary to promote the goal of an upward compatible line of FORTRANS
- providing significant extensions to FTN FORTRAN in a manner consistent with the FORTRAN language, the ANSI FORTRAN Standard and existing FORTRANs.

D.1 FORTRAN IV COMPATIBILITY WITH FTN V08.04

This section summarizes differences that may affect conversion from FTN to FORTRAN IV.

Items marked with an asterisk (*) denote differences from FTN which are common to both FORTRAN IV PLUS (F4P) and FORTRAN IV.

D.1.1 Language Differences

*1. FTN permits transfer of control to FORMAT statements, that execute as CONTINUE statements. FORTRAN IV does not, and issues compile-time error messages.

*2. FTN V07.14 provided an uncounted form for Radix-50 constants used in DATA statements. V08.04 added the counted form and promised de-support of the uncounted form. Only the counted form is provided in FORTRAN IV.

3. In FORTRAN IV, the syntax of the IMPLICIT statement is different from that required by FTN.

4. FTN provides expressions in FORMAT statements (called variable format expressions); FORTRAN IV does not.

D.1.2 Implementation Differences

*1. FTN allows both formatted and unformatted records in the same file by means of the MXDFUF subroutine call. This feature is not supported in FORTRAN IV.

*2. The FTN service subroutines PDUMP and SETPDU are not provided in FORTRAN IV. Similar but more flexible debugging output may be obtained by using WRITE statements in debug lines.

*3. The TRCLIB library for statement level execution tracing is not available in FORTRAN IV.

*4. Under FTN, Q format returns the number of characters in the input record (for example, the record length) independent of the current scan position. Under FORTRAN IV, Q format returns the number of characters remaining in the input record following the scan position. The record length may be obtained by using the Q format first in the format specification.

*5. FTN performs the following actions when opening a unit for direct access I/O:

- If the file exists; the existing file is used
- If the file does not exist; a file is created for use

In FORTRAN IV, the type of open depends on whether a READ or WRITE statement is causing the open operation. If it is a READ, then the file must already exist or an error results. If it is a WRITE, then a new file is created.

*6. The implementation of error handling in FORTRAN IV is significantly different from that in FTN. In particular, the use of error classes for controlling error handling is replaced by control over each individual error condition.

*7. FTN does not compile format specifications for use at run time, whereas FORTRAN IV does. One difference results:

> Format specifications stored in arrays are recompiled at run time each time they are used. If an H format is used in a READ statement to read data into the format itself, that data does not get copied back into the original array. Hence, it will not be available on a subsequent use of that array as a format specification. This is in accord with the ANSI FORTRAN language specification.

This consideration does not apply to format specifications defined in FORMAT statements.

*8. FTN implements the ENDFILE statement as a close operation. FORTRAN IV implements the ENDFILE statement by writing an end of file record in a file.

9. FORTRAN IV checks that the labels used in an assigned GOTO are valid labels in the program unit, but it does not check at run time whether an assigned label is in the list in the GOTO statement. FTN checks at run time.

10. The FTN /-ON compiler option switch causes two-word allocation (instead of one-word allocation) of all INTEGER and LOGICAL variables. The similar FORTRAN IV I4 (/T)

compiler option affects allocation of unspecified size INTEGER variables only; unspecified LOGICAL variables are always allocated two words in FORTRAN IV.

11. The FTN and F4P implementations of adjustable dummy arrays copy dummy argument dimension variables into an internal array descriptor upon entry to a subprogram. A subsequent assignment to a dummy argument dimension variable within that subprogram does not affect the array subscript calculation.

    FORTRAN IV does not use array descriptor blocks, but rather references dummy argument dimension variables directly during subscript calculations. Hence, an assignment to a dummy argument dimension variable will affect array subscript calculations.

12. FTN, in contrast to FORTRAN IV and F4P, does not enforce the PDP-11 Language Reference Manual's restrictions concerned with modifying the control variable and the parameters of a DO loop within the body of the loop.

13. Unlike FORTRAN IV and F4P, FTN defines named COMMON blocks in terms of named .CSECTs.

D.2 DIFFERENCES BETWEEN FORTRAN IV-PLUS AND FORTRAN IV

This section summarizes differences that may affect conversion from FORTRAN IV to FORTRAN IV PLUS.

D.2.1 Language Differences

F4P transforms FORTRAN defined function name references into a special kind of internal calling form, while FORTRAN IV does not. If a user supplies a routine of his own to replace the F4P FORTRAN defined routine (for example, writes his own SIN routine) it will generally be necessary to include EXTERNAL statements (with the user name prefixed by an asterisk, '*') to cause that routine to be referenced. This is not necessary in FORTRAN IV.

D.2.2 Implementation Differences

1. FORTRAN IV logical tests treat any non-zero bit pattern in the low-order byte of a LOGICAL variable as .TRUE., and an all-zero bit pattern as .FALSE.

   F4P tests only the highest-order bit of the value and treats a one as .TRUE. and a zero as .FALSE.

2. In FORTRAN IV, INTEGER*4 causes 32-bit allocation (4 bytes), but only 16 bits are used for computation. In F4P, INTEGER*4 causes both 32-bit allocation and 32-bit computation.

3. FORTRAN IV checks that the labels used in an assigned GOTO are valid labels in the program unit, but it does not check at run time whether an assigned label is in the list in the GOTO statement. F4P does check at run time.

4. FORTRAN IV permits an unlimited number of continuation lines. In F4P, up to 5 continuation lines are permitted by default, and up to 99 may be obtained by means of the compiler /CO switch.

5. For unformatted input/output operations, both sequential and direct access, FORTRAN IV reads/writes four bytes of data if the variable is allocated four bytes of storage. F4P does also. However, since INTEGER*4 values in FORTRAN IV generally have an undefined high-order part, they generally may not be read as INTEGER*4 values by F4P. Similarly, since FORTRAN IV and F4P logical tests are different (see item 1 above), care must be taken when interchanging logical values.

6. For random access input/output operations, FORTRAN IV takes the END= exit on an end-file condition and the ERR= exit only for hardware I/O errors. F4P ignores any END= specification and takes the ERR= exit for both.

7. In performing formatted I/O under A format, FORTRAN IV clears the high-order bit of each transferred character and discards null characters (bytes of zeroes); FORTRAN IV RSX/IAS and FORTRAN IV PLUS do neither.

D.3 RSTS/E FORTRAN IV FILE COMPATIBILITY

RSTS/E FORTRAN IV can read and write files compatible with the other language processors available on the RSTS/E operating system (BASIC-PLUS and BASIC-PLUS-2). Certain file formats, however, are not supported under FORTRAN IV. Also, care must be taken when creating files with FORTRAN if they are to be processed by a program written in another language.

D.3.1 Sequential Stream ASCII Files

FORTRAN IV, BASIC-PLUS, and BASIC-PLUS-2 share the concept of sequential stream files composed of ASCII data. In FORTRAN, this type of file is accessed with formatted READ and WRITE statements. BASIC-PLUS and BASIC-PLUS-2 process stream files through the INPUT, INPUT LINE, and PRINT statements.

A stream ASCII file consists of variable-length data records terminated by a carriage control sequence, usually CR - LF. No extraneous formatting information (such as byte counts or checksums) is included in the file.

BASIC-PLUS and BASIC-PLUS-2 programs create this type of file by default. FORTRAN will accept files structured in this fashion in formatted READ statements.

To create compatible files with the FORTRAN formatted WRITE statement, care must be taken to assure proper positioning of carriage control information. By default, FORTRAN records begin with the vertical forms control characters required for the record to be written, followed by the data record itself, and terminated by a carriage return character. For example, the statements:

```
        TYPE 100
100     FORMAT(' HELLO')
```

will create the following record:

LF HELLO RET

where

LF represents the line feed character and RET denotes the carriage return character. It is desirable to create records terminated by the carriage return-line feed sequence for compatibility. The previous example may be rewritten to achieve this:

```
        TYPE 200
200     FORMAT('+HELLO'/)
```

The '+' character suppresses the initial line feed generated by default, and the '/' record terminator causes the line feed to be generated at the end of the record, as desired.

Note that the above technique applies only to files that have been created with the 'CC' attribute in CALL ASSIGN or CALL OPEN. By default, files output to non-printing devices have carriage control translation suppressed. Such files consist of an initial line feed character, followed by records in the standard stream ASCII format (i.e., terminated by RET - LF sequences).

D.3.2 Virtual Array Files

The BASIC-PLUS language provides the capability to create files which are accessed from the program as arrays of data elements. All 'records' in the file are of a fixed length, usually one integer or floating-point value. String virtual arrays are also provided.

FORTRAN programs can read and write virtual array files in a very straightforward fashion. For example, consider the integer virtual array referenced by the following BASIC-PLUS dimension statement:

```
DIM #chan,I%(9999%)
```

To read this array file from FORTRAN, the programmer would first describe the file format with the DEFINE FILE statement as follows:

```
DEFINE FILE unit (10000,1,U,ivar)
```

Note that this statement establishes the size of each record as one word (the second argument inside parentheses), and indicates that the number of such records in the file is 10000. The file may now be accessed by the statement:

```
READ (unit'index)ivalue
```

where index represents the subscript to be used (note that FORTRAN subscripts begin at one, whereas BASIC-PLUS uses zero-origin indexing). The value of the selected element will be read into the integer variable specified by ivalue. To handle virtual arrays of floating-point values, the same format for the DEFINE FILE statement is used, replacing the size of the record in words with the value appropriate to the math package used by the BASIC-PLUS program in question. (If the two-word math package was used to create the file, the value 2 should be specified as the second argument inside parentheses; 4 is appropriate to the four-word math package.)

To handle two-dimensional virtual arrays, the DEFINE FILE statement must be coded to account for the total number of elements in the array. For example, the following BASIC-PLUS dimension statement allocates a two-dimensional integer array:

```
DIM #chan,I%(m,n)
```

where m and n specify the array dimensions (remember that indexing starts at zero in BASIC). The equivalent DEFINE FILE statement is:

```
DEFINE FILE unit ((m+1)*(n+1),1,U,ivar)
```

The expression (m+1)*(n+1) computes the number of array elements specified by the previous DIM statement. To access this array as in the following BASIC-PLUS line:

```
I1% = I%(J%,K%)
```

the FORTRAN programmer must compute the vector index as:

```
READ (unit'(J*(n+1) + K + 1))I1
```

To access a string virtual array, the DEFINE FILE statement should specify the maximum string size divided by two (as element sizes are specified in words). For example, the BASIC-PLUS statement:

```
DIM #chan,A$(100%) = 128%
```

would be equivalent to the DEFINE FILE statement:

```
DEFINE FILE unit (101,64, U, ivar)
```

The strings may be stored in the FORTRAN program in LOGICAL*1 arrays of the appropriate length. The following FORTRAN code will read one element of the virtual string array specified in the above example:

```
LOGICAL*1 STRING(128)
READ (unit'index) STRING
```

Strings stored in virtual array files are padded on the right with null characters (000 octal) to the specified record length.

For more information on string array refer to the BASIC-PLUS Language Reference Manual.

D.3.3 BASIC-PLUS Record I/O Files

BASIC-PLUS record I/O files may be accessed from FORTRAN programs using the direct access I/O facility. The DEFINE FILE statement should specify the number of words in each logical record of the file as the size. FORTRAN direct access I/O demands that all records of the file have the same length. If this is true of the record I/O file in question, the FORTRAN system will do all record blocking/deblocking for the user. If the file in question has records of several different sizes, the programmer should read the file on a block-by-block basis.

To read the file on a block-by-block basis, the DEFINE FILE statement specifies 256 as the record size. Each record is read into a 256-word array area (usually an INTEGER*2 array of 256 elements). The programmer must then find the records desired in each block by indexing through the buffer array.

When accessing values stored using the CVT%$ and CVTF$ functions, or when creating files to be read by BASIC-PLUS programs that will use the CVT$% and CVT$F functions to retrieve values, special care must be taken. The CVT%$ and CVTF$ functions store the binary values byte-reversed from the normal orientation expected by FORTRAN (and the representation in virtual array files). For example, the function call CVT%$(13%) returns the value 000,015 as octal byte values (with 000 as the low-order byte and 015 as the high-order byte). Hence, the FORTRAN programmer must reverse the bytes of each word of a value written by a BASIC-PLUS program that uses these functions. To perform the required byte-reversal operation on a floating-point value that has been read into the variable A, the following code can be used:

```
REAL*4 A
LOGICAL*1 TEMP(4),T
EQUIVALENCE (A,TEMP)
T = TEMP(1)
TEMP(1) = TEMP(2)
TEMP(2) = T
T = TEMP(3)
TEMP(3) = TEMP(4)
TEMP(4) = T
```

When writing files to be read by BASIC-PLUS, the byte reversal operation must also be performed.

APPENDIX E

THE FORTRAN IV SYSTEM SIMULATOR ($SIMRT)

E.1 $SIMRT CAPABILITIES AND RELATIONSHIP TO RT-11

$SIMRT supports the following RT-11 monitor calls:

EMT 240		.WAIT (NOP)
EMT 340		.TTYIN, .TTINR
EMT 341		.TTYOUT, .TTOUR
EMT 342		.DSTATUS (only for TT:)
EMT 346		.LOCK (NOP)
EMT 347		.UNLOCK (NOP)
EMT 350		.EXIT
EMT 351		.PRINT
EMT 353		.QSET (NOP)
EMT 354		.RCTRLO
EMT 374	Subcode 8.	.DATE(returns 0)
EMT 375	Subcode 0,	.WAIT (NOP)
	Subcode 3,	.TRPSET
	Subcode 16,	.GTJB (no channel table adrs)
	Subcode 24,	.SFPA
	Subcode 27,	.CNTXSW (NOP)

Any others create an error condition and a printed message.

E.2 $SIMRT TERMINAL HANDLING

$SIMRT terminal I/O is fully interrupt-driven and ring-buffered.

The following features are supported:

CTRL-S, CTRL-Q (XON, XOFF) -- "SET TT PAGE" mode

CTRL-O to cancel terminal output

CTRL-U to cancel input line

Rubout to delete previous character (echoes backslashes)

CTRL-C to abort execution

Conversion of TABs to spaces on output

Output of altmode (escape) as "$"

Output of CTRL characters as "^char"

Support of lowercase input if done with .ASECT, for example:

```
.ASECT
. = 44 ;for JSW
.WORD   040000
```

Support of RT-11 style fill character and count with .ASECT, i.e.:

```
.ASECT
. = 56  ;for fill char and count
.BYTE   015,004    ;four nulls after carriage return
```

The TT$SPC mode (single-character mode) is not supported.

E.3 SYSLIB CALLS UNDER $SIMRT

Routine Name	Valid?	Routine Name	Valid?
CHAIN	No	ISPY	No
CLOSEC	No	ITIMER	No
CONCAT	Yes	ITLOCK	No
CVTTIM.	Note #1	ITTINR	Yes
DEVICE	No	ITTOUR	Yes
GETSTR	Yes	ITWAIT	No
GTIM	No	IUNTIL	No
GTJB	Note #2	IWAIT	Note #2
GTLIN	No	IWRITE,W,C,F	No
IADDR	Yes	JADD	Yes
IASIGN	Note #2	JCMP	Yes
ICDFN	No	JDIV	Yes
ICHCPY	No	JAFIX	Yes
ICMKT	No	JDFIX	Yes
ICSI	No	IDJFLT,DJFLT	Yes
ICSTAT	No	IAJFLT,AJFLT	Yes
IDELET	No	JICVT	Yes
IDSTAT	Note #2	JJCVT	Yes
IENTER	No	JMOV	Yes
IFETCH	No	JMUL	Yes
IFREEC	Note #2	JSUB	Yes
IGETC	Note #2	JTIME	Note #1
IGETSP	Yes	LEN	Yes
IJCVT	Yes	LOCK	Note #2
ILUN	Note #2	LOOKUP	No
INDEX	Yes	MRKT	No
INSERT	Yes	MTATCH	No
INTSET	No	MTDTCH	No
IPEEK	Yes	MTGET	No
IPEEKB	Yes	MTIN	No
IPOKE	Yes	MTOUT	No
IPOKEB	Yes	MTPRNT	No

(continued on next page)

Routine Name	Valid?	Routine Name	Valid?
IQSET	Note #2	MTRCTO	No
IRAD50	Yes	MTSET	No
IRCVD,W,C,F	No	MWAIT	No
IREAD,W,C,F	No	PRINT	Yes
IRENAM	No	PURGE	No
IREOPN	No	PUTSTR	Yes
ISAVES	No	R50ASC	Yes
ISCHED	No	RAD50	Yes
ISDAT,W,C,F	No	RCHAIN	Note #2
ISLEEP	No	RCTRLO	Yes
ISPFN,W,C,F	No	REPEAT	Yes

Routine Name	Valid?
RESUME	No
SCCA	No
SCOMP	Yes
SCOPY	Yes
SECNDS	No
SETCMD	No
STRPAD	Yes
SUBSTR	Yes
SUSPND	No
TIMASC	Note #1
TIME	No
TRANSL	Yes
TRIM	Yes
UNLOCK	Note #2
VERIFY	Yes

Note #1: Will operate if the following user routine is added:

```
        .GLOBL  $GVAL
$GVAL:  CLR     R0      (if 60-cycle clock)
        MOV     #40,R0  (if 50-cycle clock)
        RTS     PC
```

(This routine should be in .PSECT USER$I)

Note #2: Operates correctly, but is not useful in the $SIMRT environment.

E.4 MODIFYING SIMRT

To modify the stand-alone FORTRAN support module, SIMRT, obtain a copy of SIMRT.MAC and FRT.MAC from your binary distribution medium according to the procedures described in the RT-11 Installation Guide/Release Notes or the RSTS/E Installation Guide/Release Notes, as appropriate. If you wish to modify SIMRT.MAC, for example, to support new devices, be certain that MACRO.SAV, and SYSMAC.SML are on your system volume. The command procedures (provided below) will not work without them. Next, assemble SIMRT.MAC by typing the following command:

```
.MACRO/LIST:SIMRT/OBJECT:UNI FRT+SIMRT
```

This command produces a listing file named SIMRT.LST and an object file named UNI.OBJ.

Next, place the new stand-alone support module in the FORTRAN OTS Library using the appropriate command procedure:

If your FORTRAN OTS Library is in SY6LIB, then proceed as follows:

RT-11

```
.R LIBR
*SY6LIB[-1]=SY6LIB,UNI/U/G
Global?  $ERRS
Global?  $ERRTB
Global?  $OVRH
Global?  RET
*^C

.
```

RSTS/E

```
.R LIBR
*SYSLIB[-1]=SY6LIB,UNI/U
*^C

.
```

If your FORTRAN OTS Library is in FORLIB, however, then proceed as follows:

RT-11

```
.R LIBR
*FORLIB[-1]=FORLIB,UNI/U/G
Global?  $ERRS
Global?  $ERRTB
Global?  RET
*^C

.
```

RSTS/E

```
.R LIBR
*FORLIB[-1]=FORLIB,UNI/U
*^C

.
```

The new version of stand-alone FORTRAN support is now part of your FORTRAN OTS Library.

NOTE

You must assume responsibility for changes, such as those shown above, that you make to SIMRT.MAC or to FRT.MAC. Such changes are not supported by DIGITAL.

A

Absolute binary format (LDA), 1-18, 1-22
ACCEPT statement, 3-6
Argument,
  passing using COMMON, 4-2
  transmission of, 2-9
Arguments, ASSIGN routine, B-2
Arrays,
  multi-dimensional, 2-12, 2-13
    use of, 4-3
  passed to subprograms, 2-12
    optimizing, 4-2
  storage space, 2-12
  two-dimensional, 2-13
  vector maps, 2-13
  vectored, 2-12 to 2-13
  table storage reduction, 4-2
  virtual, 2-4 to 2-8
  zeroing large, 4-6
ASCII characters, 3-7
  RADIX-50 equivalents,
    table of, A-4
  records, 3-6
  transfer of files, 3-7
Assembler, MACRO, 1-17
Assembly language subroutines, 1-27, 2-10
ASSIGN command, 3-5
ASSIGN subroutine, 1-25, B-1, B-2
Assigned GOTO, 4-4

B

BACKSPACE statement, 3-7
BASIC-PLUS files, D-6
Binary,
  conveying data, 3-7
  exponents, A-1
  output file (binout), 1-20
  records, 3-6
Blank COMMON, 1-15
Blank records, 3-4
Blanks, imbedded in command string, 1-7
BLOCK DATA statement, 2-1
Blocks, COMMON, 1-15, 1-25, 1-26, 2-17
Buffered I/O, 3-8
Buffers, internal, B-3

C

Carriage control, 3-7, B-3, B-4
Carriage return, 3-5
CALL statements, 2-10
Calling program,
  passing control to, 2-9
CCL (Concise Command Language), 5-1 to 5-2
  restricted switches, 5-2
Character,
  ASCII, A-5
  line feed, 3-4, 3-7
  RADIX-50, A-4
CLOSE subroutine, B-1, B-6
Code,
  in-line, 1-2, 1-3, 2-1, 2-2
  object, 1-15, 2-1
  threaded, 1-2, 1-3, 2-1, 2-2
Code selection, 1-3
Command string, 1-1, 1-7
  compiler, 1-1, 5-1
  imbedded blanks in, 1-7
  linker, 1-1, 1-17
  switch options in, 1-9
COMMON,
  blank, 1-15
  block, 1-15, 1-25, 1-26, 2-17
  data initialized, 1-26
  DATA statements, 1-26, 1-27, 3-1
  using to pass arguments, 4-7
  variables, initialization of, 3-4
Compatibility, language, D-1
Compiler,
  command sequence, 1-1
  command string, 1-1, 1-7
  error diagnostics, C-1
    fatal, C-13
    warning, C-12
  errors,
    initial phase, C-2
    secondary phase, C-4
  generated code, 1-2, 1-3, 2-1, 2-2
  input files, 1-7
  listings, 1-8, 1-11
  memory requirements, 1-16
  object code, 1-15
  options, 1-2
  output files, 1-7
  referencing library instruction, 2-1
  running the, 1-7
  switches, 1-9
Compiling a program, steps in, 1-1
  example of,
    RSTS/E, 1-8
    RT-11, 1-7

Compiling a program, steps in, (Cont.)
increasing effectiveness of, 4-1
sample listing, 1-13
statistics listing, 1-14
COMPLEX format, A-2
COMPLEX*8, 4-4
Concise Command Language (CCL), 5-1 to 5-2
Conditional (/D) compilation switch, 1-31
Configurations and valid object code options, 1-2
Continued lines, 3-4
Core image file LOAD.SAV, 1-20

D

/D (Comment),
conditional compilation, 1-31
Data,
ASCII, 3-6, 3-7, A-4
binary, 3-7
conversions, 4-4
representation, A-1
DATA statement, 1-26, 3-1, 3-4, 4-7
Data types, 4-4
DATE subroutine, B-1, B-6
Debugging FORTRAN IV, 1-31
DECODE statement, 3-4
Default devices and file names, 3-5
Default memory, 1-7, 1-8
DEFINE FILE statement, 3-6
Device,
changing default, 3-5
code, 1-3
random access, 1-20
specifications, 1-4
Diagnostics, error,
compiler, C-1
fatal, C-13
warning, C-12
Object Time System, C-14
DIMENSION statement, 2-6
Direct access
files, 3-6
input/output, 3-8
Directory, user's, 1-7
Dividing a program, 1-25
DO loops,
increment parameter in, 4-5
nesting of, 4-6
Dollar sign ($),
format separator, 3-7
DOUBLE PRECISION,
data type, 4-4
format, A-2

E

ENCODE statement, 3-7
END statement, 3-6
ENDFILE record, 3-8
Equivalents, ASCII/Radix-50, A-5
ERR= parameter, 3-1, 3-7
Errors,
compiler, C-1
fatal, C-13
initial phase, C-2
secondary phase, C-4
warning, C-12
locating run-time, 2-17
message format, C-1
Object Time System, C-14
run-time, 2-17, 3-7
Execution procedures, 1-30
on a satellite machine, 1-28
EXIT subroutine, B-1, B-7
Exponents, binary, A-1
.ext (extension), 1-3, 1-6
External subroutine,
assigned GOTO, 4-4

F

Fatal errors, compiler, C-13
File,
ASCII, 3-7
sequential stream, D-4
transfer of, 3-7
BASIC-PLUS, D-6
binary (binout), 1-22
compatibility, D-4
compiler output, 1-7
LOAD.SAV, 1-17, 3-6
IAM, D-7
input, 1-3, 1-6, 1-7
library, in command string, 1-24
load map (mapout), 1-17
locating, 1-7
mode,
'NEW', B-3, B-4
'OLD', B-3, B-4
'RDO', B-3, B-4
'SCR', B-3, B-4
multiple input, 1-7
specification, load map, 1-17
VIRTUAL array, 2-4, D-5
File name,
default assignments, 3-5
extensions, 1-6
specifications, 1-3, 1-7
FIND statement, 3-6
Floating-point formats, A-1
FORLIB (FORTRAN IV System Library), 1-7

Format separator ($), 3-7
FORMAT statement, 3-7, 4-4
Formatted input/output, 3-7, 4-3
FORTRAN data representation, A-1
FORTRAN IV,
  compatibility,
    RSTS/E file, D-4
    with FTN V08.04, D-1
  debugging, 1-29
  differences (IV and IV-PLUS), D-3
  operating environment, 2-1
  stand-alone, 1-27
  system library (FORLIB), 1-17
  using the system, 1-1
FORTRAN statement, 3-1
Four-word floating point format, A-2
FPU (floating point unit) status, 2-11, 2-12
Fractions, A-1
Function,
  mapping, 2-12
  statement, 4-2
  subprogram, 2-11

G

Generated code, listing, 1-15
GOTO statements, 4-4

H

Handler, overlay, 1-26
Hardware registers, 2-11
Hollerith format, A-3

I

IAM files (Indexed Access Method), D-7
IDATE subroutine, B-1, B-7
Increment parameter in DO loops, 4-5
Initial phase errors, C-2
In-line code, 1-2, 1-3, 2-1, 2-2
Input files,
  assumed extension, 1-6
  file name specification, 1-3
  multiple, 1-7
Input/output,
  direct-access, 3-8
  efficient operation of, 3-8
  'FOR' (formatted), B-5
  format of, 3-7
  formatted routines, 3-7, 4-4
  'RAN' (random-access), B-5
Input/output, (Cont.)
  'UNF' (unformatted), B-5
  unformatted routines, 3-7
INTEGER mode,
  calculations in, 4-6
  data type, A-1, 4-4
Internal buffers, B-3
Internal sequence numbers, 1-15
Internal subroutines, 4-2
ISN (internal statement numbers), 2-5

J

Job, virtual, 1-27, 1-28

L

LDA output file,
  absolute binary format, 1-6, 1-18, 1-20, 1-22, 1-28
  save image format, 1-17
Library,
  creation, 1-24
  file, 1-24
  modification of, 1-25
  routines, 2-2
  subroutine summary, B-1
  system, 1-7
Line feed character, 3-6
Linker (LINK), 1-17, 2-16, 5-1
  command, 1-17, 1-20
  command string, 1-1
  linking procedures, 1-17
    for subprograms, 2-9
  memory value, 1-23
  object module, 1-17
  overlay feature, 1-25
  root segment, 1-25
  RSTS/E linking, 1-20
  RT-11 linking, 1-17
  switches, 1-21
Listing,
  compilation statistics, 1-14
  expanded, 1-9
  formats, 1-11
  generated code, 1-5
  optional sections, 1-9
  options, 1-15
  sample compilation, 1-13
  source, 1-15
  storage map, 1-15
LOAD.LDA command sequence, 1-28
Load map (mapout) file, 1-17
  specification, 1-17, 1-21
LOAD.SAV core image file, 1-21
Locating a file, 1-7
LOGICAL (LOGICAL*4) format, A-3

LOGICAL*1 format, A-3
Logical units,
  assignments, 3-5
  maximum open (NLCHN), 1-15
  numbers (LUNs), 3-5
Loop,
  calculations outside, 4-6
  calculations within, 4-7
LRECL (maximum record length), 1-15

M

MACRO assembler, 1-17
Maps,
  array vector, 2-12
  load (mapout) file, 1-21
  mapping function, 2-12
  subprogram vector, 2-12
Memory, 1-16, 2-18, 2-19
  read-only, (ROM), 1-27, 2-16, 2-17
  VIRTUAL, 2-5
Mode,
  calculation in INTEGER, 4-4
  file,
    'NEW' (new file), B-3, B-4
    'OLD' (existing file), B-3, B-4
    'RDO' (read only), B-3, B-4
    'SCR' (temporary), B-3, B-4
  mixed, 3-8, 4-4
Modules,
  linker object, 1-17
  system simulator ($SIMRT), 1-28
Multi-buffering, B-3
Multi-dimensional arrays, 2-12, 2-13, 4-3

N

Nesting of DO loops, 4-6
NLCHN (maximum open logical units), 1-15
Numbers, A-1

O

Object,
  program efficiency, 4-1
  time format, 4-3
Object code, 1-2, 1-15, 2-1
Object modules, linker, 1-17
Object Time System (OTS), 2-1, C-13
OPEN statement, restrictions, B-4
OPEN subroutine, B-1, B-3, B-4
Operations (*2, **2), 4-5
Optimizer, effective use of, 4-1
Optional listing sections, 1-9, 1-15
Options, compiler, 1-2, 1-3, 1-9
Options, linker,
  RSTS/E, 1-21
  RT-11, 1-17
OTS (Object Time System), 2-1
  error diagnostics, C-13
Output file,
  absolute binary format, 1-22
  compiler, 1-12
  default extension, 1-6
  name specification, 1-3
Output format, 3-7
  see also Input/output
Overlay, program, 1-24 to 1-27, 2-16, 5-2

P

Parameter, ERR=, 3-1, 3-7
Parity bit, ASCII, 3-7
PAUSE statement, 3-4, 3-6, 3-8
PRINT statement, 3-6
Program,
  calling, 2-10
  compiled,
    protection code, 1-6
  division, 1-15
  main, placement of, 1-20
  overlay, 2-16
  preparing source, 1-1
  steps in executing, 1-1
  storage required, 1-9
  system, invoking RSTS/E, 5-1
Program Logical Address Space (PLAS), use with VIRTUAL, 2-5
Program sections (PSECT), 1-15, 2-14 to 2-17, 5-2
  assembly language in, 2-16
  attributes, 2-14
  COMMON in, 2-15
  compiled output, 2-14
  MACRO in, 2-17
  ROMs, building in, 2-16
  USR in, 2-16
Program termination,
  fatal error condition, C-13
Programming techniques,
  efficient, 4-5
  for division in programs, 4-7
  minimizing execution space, 4-4
Project, programmer number [p,pn], 1-7

Protection code (<prot>), file, 1-6
PSECT
see Program sections

R

RADIX-50 format, A-4
RANDU, RAN subroutine, B-1, B-5, B-8
format of, B-8
READ statement, 3-5, 3-6
REAL format, A-2
REAL*4 data type, 4-4
Record,
ASCII, 3-7
binary, 3-7
changing length, 3-6
maximum length, 1-15, 3-6
formatted, 3-7
Region, overlay, 1-25
Register,
assignments, 2-11
hardware, 2-11
subprogram usage, 2-11
Register 5 (R5), 2-10
Relocatable code, specifying address of, 1-20, 1-22
REWIND statement, 3-8
ROM (read-only memory), 1-27 2-16, 2-17
Root segment, linker, 1-25
RSTS/E,
compilation and memory requirements, 1-16
linking procedure, 1-20
RT-11,
compilation and memory requirements, 1-16
linking procedure, 1-17
RUN command, 1-1, 1-8
Run-time errors,
interception of, 3-7
locating, 2-17
Run-time memory, 2-18, 2-19

S

Satallite machine, 1-27
ROM applications, 1-27
Secondary phase errors,
compiler, C-4
summary, C-4
Segment,
memory, 2-18
overlay, 1-25
Sequential stream ASCII files, D-4
SETERR subroutine, B-1, B-8, C-13
$SIMRT, E-1
Size, overlay, 1-26
Source listing, 1-14
Source program, preparing a, 1-1
Specifications,
device, 1-4
file name, 1-3
Stack, hardware, 1-21, 1-23
contents of, 2-11
Stand-alone FORTRAN programs, 1-28
Statement, FORTRAN, 3-1
functions, 4-2
Statements,
ACCEPT, 3-6
CALL, 2-10
CLOSE, B-1, B-6
DATA, 3-1
DECODE, 3-4
DEFINE FILE, 3-6
ENCODE, 3-7
END, 3-7
FIND, 3-6
FORMAT, 3-5, 4-3
GOTO, 4-5
OPEN, 3-1
PAUSE, 3-4, 3-6, 3-8
PRINT, 3-6
READ, 3-6, 3-7
STOP, 3-4, 3-5
TYPE, 3-6
WRITE, 3-7
STOP statement, 3-4, 3-5
Storage map listing, 1-15
String, command, 1-7
Structure, overlay, 1-26, 5-2
Subprograms,
arguments passed to, 4-2
arrays passed to, 2-12
control passed to, 2-9
function, 2-11
linkage, 2-9
register usage, 2-11
vector maps, 2-13
Subroutines, assembly language, 1-17
internal, 4-2
summary, B-1
Subroutines, library, B-1
ASSIGN, B-2
CLOSE, B-6
DATE, B-6
ERRSNS, B-9
ERRTST, B-8
EXIT, B-7
IDATE, B-7
OPEN, B-3
RANDU, RAN, B-8

Subroutines, library, (Cont.)
SETERR, B-8, C-13
USEREX, B-7
Switch options, 1-2
Switches, CCL command restriction, 5-2
Switches, compiler, 1-9
Switches, linker, 1-17, 1-21
System library, 1-7
System simulator, E-1

T

Threaded code, 1-2, 1-3, 2-1, 2-2
Traceback feature, 2-17
Transferring ASCII files, 3-7
Translation of carriage control, 3-7, B-3
Two-dimensional array, 2-12
Two-word floating point format, A-2
TYPE statement, 3-6

U

Unformatted input/output, 3-7
USEREX subroutine, B-1, B-7
User's directory, 1-7
USR (User Service Routine), 2-18

V

Variables, COMMON, 3-4
Vectored arrays, 2-12 to 2-13
table storage reduction, 4-2
VIRNP.OBJ, 2-5
VIRP.OBJ, 2-5
VIRTUAL array, 2-4 to 2-8
VIRTUAL array files, D-5
Virtual job, 1-27, 1-28
VIRTUAL memory, 2-5
VIRTUAL statement, 2-6
.VSECT, 2-4

W

Warning diagnostics, compiler, C-12
WRITE statement, 3-7

Z

Zeroing a large array, 4-6

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

Did you find errors in this manual? If so, specify the error and the page number.

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

Please indicate the type of reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify)_____

Name_____ Date_____

Organization_____

Street_____

City_____ State_____ Zip Code_____
or
Country

Please cut along this line.

Do Not Tear - Fold Here and Tape

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

BSSG PUBLICATIONS TW/A14
DIGITAL EQUIPMENT CORPORATION
1925 ANDOVER STREET
TEWKSBURY, MASSACHUSETTS 01876

Do Not Tear - Fold Here

Cut Along Dotted Line